Software Specifications

System Control CSCI
Redundancy Management CSC
Thor 3.0
DP-3

Checkout and Launch Control System (CLCS)

84K00570-130

Approval:

Chief, System Software Division	Date		Date
	Date		Date
	Date		Date

Prepared By:

Michael Symes 06/19/1998

NOTE: See "**Supporting Document Note**" on following page

REVISION HISTORY

REV	DESCRIPTION	DATE
Basic	Promoted per approval by Design Panel. ljp	5/15/98
A	Promoted per approval by Design Panel. ljp	7/7/98

Supporting Document Note:

Acronyms and definitions of many common CLCS terms may be found in the following documents: CLCS Acronyms 84K00240 and CLCS Project Glossary 84K00250.

Table of Contents

1.1 Redundancy Management Design Specification

As discussed above, the Redundancy Management CSC consists of 6 major parts. Each of these parts is discussed separately below. For Thor 3.0, only the sections regarding Subsystem Integrity are the focus of this document. In an effort to convert to the current CLCS DP3 template, many of the sections within this document outside the scope of this discussion simply state TBD. It is not the intention of this document to present that information. Information relevant to those sections outside the scope of Thor 3.0 will be provided during the Atlas delivery.

1.1.1 Set Integrity

Set Integrity is not provided in Thor.

1.1.2 System Integrity

System Integrity's role is to monitor data about the current state of the system and its parts, and based on that information determine whether the system is healthy. *When a failure is detected, SI determines the appropriate recovery action*. SI consists of four parts: the first is resident on the Master CCP and is the part generally referred to as SI or SI Master. This part of SI contains the logic to determine that a failure has occurred and what recovery action is appropriate. The second is a Health Count Monitor that resides on the DDP. A key factor in the health of a computer is its ability to periodically send a health counter FD. This FD is initially received at the DDP, then distributed to all other computers. The latency of data to the CCP can be as much as 2 SSRs. In order to meet performance requirements, detection of a missed health counter must take place on the DDP prior to the distribution. The Health Count Monitor executes periodically to ensure that all health counters have been received. If any have not been received, the Health Count Monitor issues a Missed Health Counter SEC to SI on the Master CCP. The third part is referred to as Mini-SI and resides with the Master SCT prior to initialization of the Master CCP. In order to change the set configuration prior to initialization of the Master CCP and DDP, the Master SCT must be modifiable prior to initialization of the Master CCP. The Redundancy Management initialization process allows the Master SCT to execute on any computer. Because SI supplies all modifications to the SCT, a small version of SI must be able to run on other computers as well. This is a simplified SI. It does no analysis or recovery, but is able to process CCWS initiated configuration changes and is able to process SECs from other computers. Finally, a small layer of SI must reside on the Master CCWS to process API invocations from the System Status Viewer and other software able to request SCT changes. This API layer receives the procedure call and forwards the request to either Mini-SI or SI at the Master CCP, whichever is active.

1.1.2.1 System Integrity Detailed Data Flow

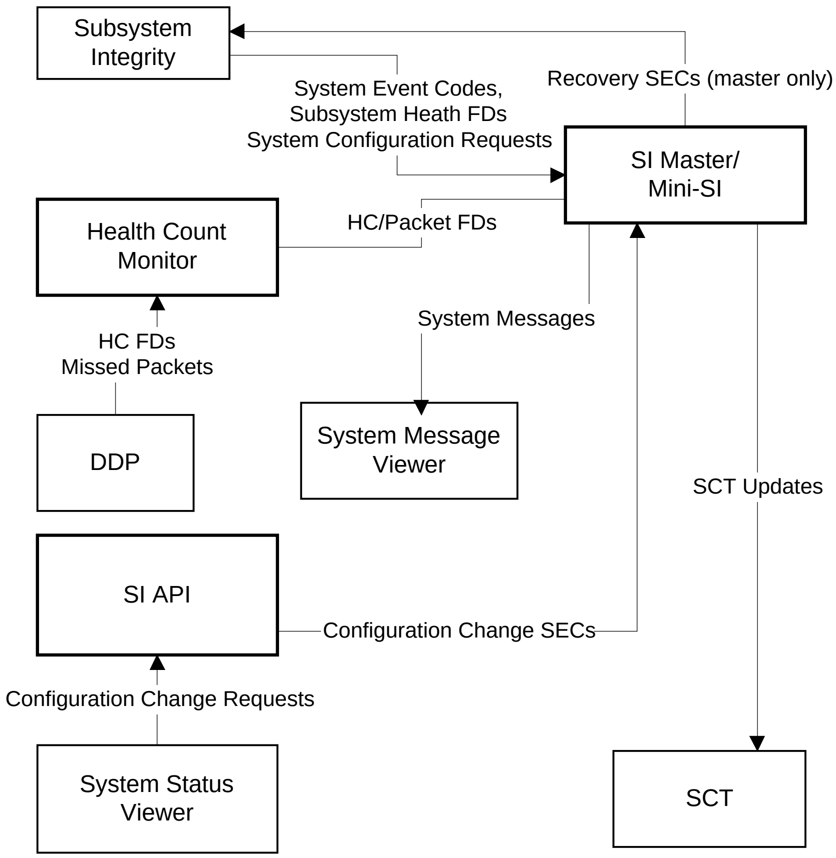


Figure 1. SI Detailed Data Flow

Inputs to SI Master or Mini-SI:

- System Event Codes: These event codes are messages generated by Subsystem Integrity that serve as the primary information source for SI decisions. There are two general classes of these SECs:
- State Changes: As a subsystem completes initialization, it passes through a series of states (In configuration, Loaded, Communicating, Go). As it enters each state, a System Event Code is generated to inform SI that it has achieved the state. SI then incorporates that information into the master SCT. If a subsystem fails, or is directed to terminate, it also sends SECs, if able to notify SI of the movement into a lower state. Because many failure modes will not allow these SECs to be issued, SI does not rely on them for information, but uses them if available.
- Error Reports: These error reports are generated by the Health Count Monitor, and are used to inform Master SI that the Health Count Monitor has detected either a missing HC, or a HC that has an unexpected value.
- Subsystem Health FDs: These FDs contain specific health information data, usually performance related that can be used by SI to infer information about the health of the box. Thor constraint: These FDs are not used in health determination, but error related FDs are relayed as System Messages to allow notification of unexpected events.
- System Configuration Requests: The SI API generates the requests to modify the configuration. This typically occurs prior to initialization of the test set, but also occurs when manually reconfiguring around a failed piece of hardware, or to activate hardware that was allocated to the test set, but not initially used. SI does some sanity checking to ensure the request is valid, then updates the SCT to reflect the requested change.

Outputs from SI Master or Mini-SI:

- SCT Updates: Any modifications to the system configuration are recorded through calls to the SCT interface. By definition, the Master SCT is the copy of the SCT that resides on the same computer as Mini-SI or SI Master. The Master SCT then propagates these updates to all copies of the SCT.

Outputs from SI Master:

- *Recovery System Event Codes: These messages direct subsystems to switch, terminate or take other recovery actions.*
- System Messages: These user messages allow the operations team to track events in the system.

Inputs to the Health Count Monitor:

- HC FDs: These FDs are generated cyclically by Subsystem Integrity in all subsystems in all roles (active, standby, spare). An incremented health counter indicates the subsystem believes it is healthy. A decremented health counter indicates the subsystem believes it is not healthy. An unchanged health counter indicates that an update was not received. Any unexpected value results in a SEC to SI Master to notify it of the failure.
- Missed Packet Notification: Health Count Monitor provides an API to the DDP CSCI so that it can notify SI of a missed packet. This local procedure call on the part of the DDP CSCI results in an SEC being sent by Health Count Monitor to Master SI.

Outputs from Health Count Monitor:

- System Event Codes: These messages to Master SI inform it that a health counter was not received or contained an unexpected value, or that an expected packet was not received.

Inputs to SI API:

- Requests for configuration changes: These invocations of the local API are made by the System Status Viewer as part of the initial configuration of the test set. It allows re-allocating subsystems to resources or activating resources that were allocated to the test set, but are not yet in use by the test set. It also provides an interface of users to manually invoke switchover or termination.

Outputs from SI API:

- Configuration Change Requests: These messages relay the requests made by the users to either SI Master or Mini-SI..

1.1.2.1.1 Checking Health Counters

Health Counters are received for each computer in the system. The exact method of receipt varies by computer type. All Health Counter checking takes place on the DDP.

1.1.2.1.1.1 Gateways

The DDP expects data periodically from the gateways, and notifies SI on the Master CCP via a Missed Packet System Event Code when a packet is not received. Each of these packets should contain a Health Counter FD in them, Health Count Monitor ensures that they are received and incremented.

1.1.2.1.1.2 CCPs and DDPs

Health Counter FDs are generated for the CCPs and DDPs. The Data Distribution SSR process invokes the HC Check software once each iteration. This software checks the current values of CCP and DDP FDs against expected values. Any discrepancies are reported to SI on the master CCP via the HC Did Not Increment or HC Decremented SECs.

The periodic check under an existing process minimizes the CPU required for this high frequency operation. However, there is no apparent difference between a missed CCP/DDP HC and a CCP/DDP HC that was sent, but did not increment. The design of HC generation is such that sending a HC that has not changed should never occur.

1.1.2.1.1.3 CCWSs

CCWS HC monitoring also occurs at the DDP, and uses the same general techniques as the CCPs and DDPs. However, the health check for CCWSs is at the DSR rather than at the SSR. Given the lower frequency and lower priority, this check is in its own process.

1.1.2.1.2 Handling System Event Codes

SI Master receives SECs directly from Reliable Messaging through a local queue. The System Event Code handler is a standalone process that suspends on the Reliable Messaging queue. When a SEC is received, it is processed immediately. Because there is a tight deadline on this aperiodic process, its priority is equivalent to that of a 10ms cyclic process. A SEC that notifies SI of a state change is used to update the SCT. SI generates a System Message to notify the user community of the state change. Error reports also generate System Messages.
Significant additional handling of events and error recovery provided in Atlas.

1.1.2.2 System Integrity Context Diagram

TBD

1.1.2.3 System Integrity State Definition and State Transition Diagram

TBD

1.1.2.4 System Integrity Unique Algorithm Design

TBD

1.1.2.5 System Integrity Development Tools

TBD

1.1.2.6 System Integrity External Interfaces

System Integrity generates no System Messages. Subsystem state change messages are generated out of the SCT.

1.1.2.7 System Integrity Data Dictionary

TBD

1.1.2.8 System Integrity Message Formats

TBD

1.1.2.9 System Integrity Display Formats

System Integrity owns no displays

1.1.2.10 System Integrity Input Formats

System Integrity receives the following System Event Codes:

Table I. System Event Codes

SEC Number	Name	Source	Destination
256	Subsystem Loaded	SSI	SI (Master SCT)
257	Subsystem Communicating	SSI	SI (Master SCT)
258	Subsystem Go	SSI	SI (Master SCT)
259	Subsystem NoGo	SSI	SI (Master SCT)
260	Not Communicating		
261	Subsystem Not Loaded	SSI	SI (Master SCT)
393	*Subsystem running ORT*	*SSI*	*SI*
394	Subsystem not running ORT	SSI	SI (Master SCT)
395	No Packet Received from GW	SI-DDP	SI-CCP
396	Standby GSE detected no poll from Active GSE	GSEnS	*SI*
397	GSE reports no response from bus	GSEnA	*SI*
398	HC not Incremented	SI-DDP	SI-CCP
399	HC has Decremented	SI-DDP	SI-CCP
Additional - to be defined during implementation	User requested configuration changes	SI-API	SI-CCP

1.1.2.11 System Integrity Recorded Data

System Integrity records no data other than that automatically recorded through RM.

1.1.2.12 System Integrity Local Storage Requirements and Formats

TBD

1.1.2.13 System Integrity Printer Formats

System Integrity produces no printed data

1.1.2.14 System Integrity Interprocess Communications (C-to-C Communications)

All SI interprocess communications are through SECs as defined above.

1.1.2.15 System Integrity External Interface Calls (e.g., API Calling Formats)

An API is provided to allow the System Status Viewer to request configuration changes. This API is available at http://www-clcs/project/syscontrol/redman/ClassList.html

1.1.2.16 System Integrity Table Formats

System Integrity uses no files.

1.1.3 Subsystem Integrity

Subsystem Integrity (SSI) executes in each CCP, DDP, CCWS and monitors the health of the subsystem. A similar SSI also resides in each of the gateways, however it is not part of this CSC and is therefore not represented within this document. SSI reports this health, and any subsystem state changes to System Integrity. The primary SSI to SI communication path is the Health Counter FD. SSI analyzes data available to it about the subsystem health, then sends either a healthy health counter (incremented) or an unhealthy health counter (decremented). *SI also responds to control directives issued by SI.*

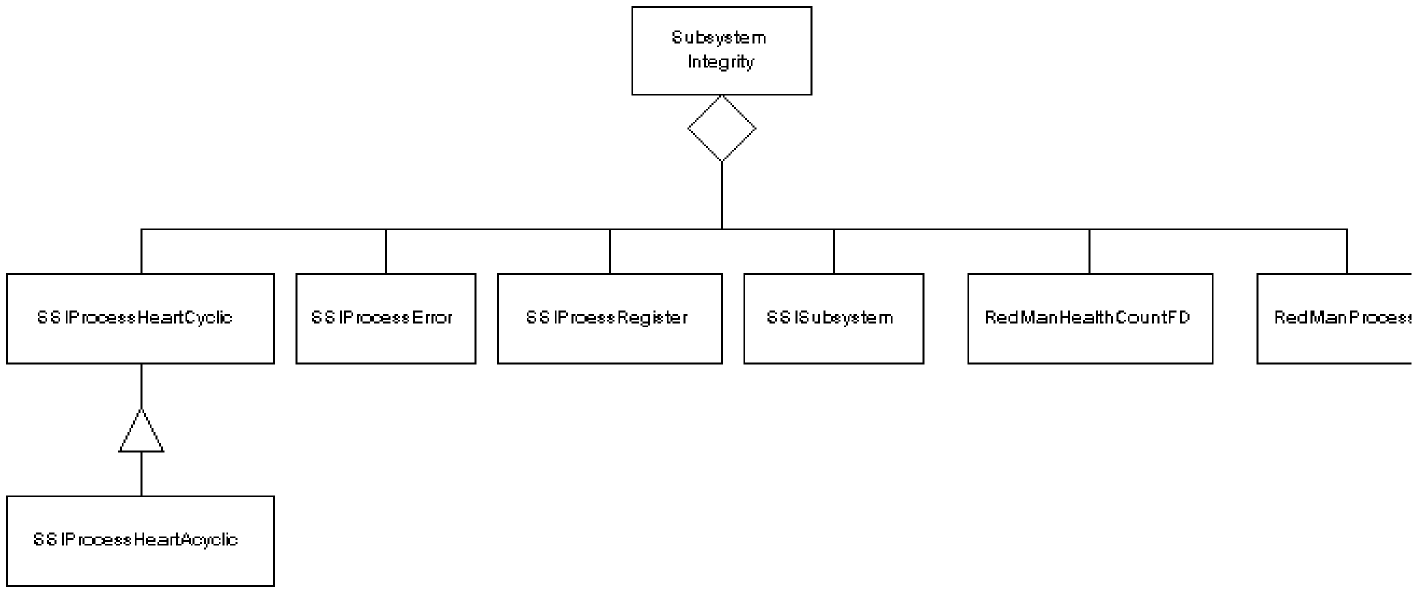


Figure 2. Subsystem Integrity Classes

Subsystem Integrity is composed of the following classes:

- SSIProcessHeartCyclic: This class is the local health counter that ensures each process is executing correctly.
- SSIProcessHeartAcyclic: This class is the local health counter that ensures each process is executing correctly.
- SSIProcessError: This class provides the interface and processing for individual errors that occur during process execution.
- SSIProcessRegister: This class provides the registration capabilities for a process to be formally registered. It contains data or pointers to data concerning the registration status of a process.
- SSISubsystem: This class is an extension of the readable class used by the SCT. It provides a mechanism for the SSI to track its own state, and for OPS/CM to request the transition to the Loaded, Comm, and Go states.
- RedManHealthCountFD: This class is for the Application Health Counter FD. It provides a mechanism for SSI to report its health, as determined by SSI, to SI.
- RedManProcess: This class is also an extension of the class defined by the SCT. It contains data or pointers to data concerning the health and criticality of individual processes in the subsystem.

1.1.3.1 Subsystem Integrity Detailed Data Flow

Startup interactions between SI and the Ops/CM Server are shown in Figure 3. Ops CM Server initiates the processes in the subsystem. Early in the startup sequence, Ops/CM loads the Redundancy Management Software. As part of its loading, Redundancy Management loads the SCT into shared memory and activates all Redundancy Management processes including a high priority 100 Hz process on the CCP and the DDP or 10 Hz process on the CCWS that monitors its subsystem health. Control is then returned to Ops CM Manager. Ops/CM Manager continues registering and loading the other System Software. When all software has been registered and loaded, Ops/CM Manager invokes the Subsystem.Load method on the My subsystem from the SCT. This marks local data, not visible through the API, as OPS/CM Loaded. At its next cycle, the SSI process checks the state and determines it has changed. A state transition is requested by issuing the Subsystem Loaded SEC to SI. Assuming SI agrees with the request, the SCT will be updated. SSI then begins sending the Health Counter FD as part of its 100 Hz process. This should eventually be reflected in the SCT as a transition to the Communicating state. At some later point, when all System Software is executing, the same sequence of events occurs to move the subsystem to the Go state.

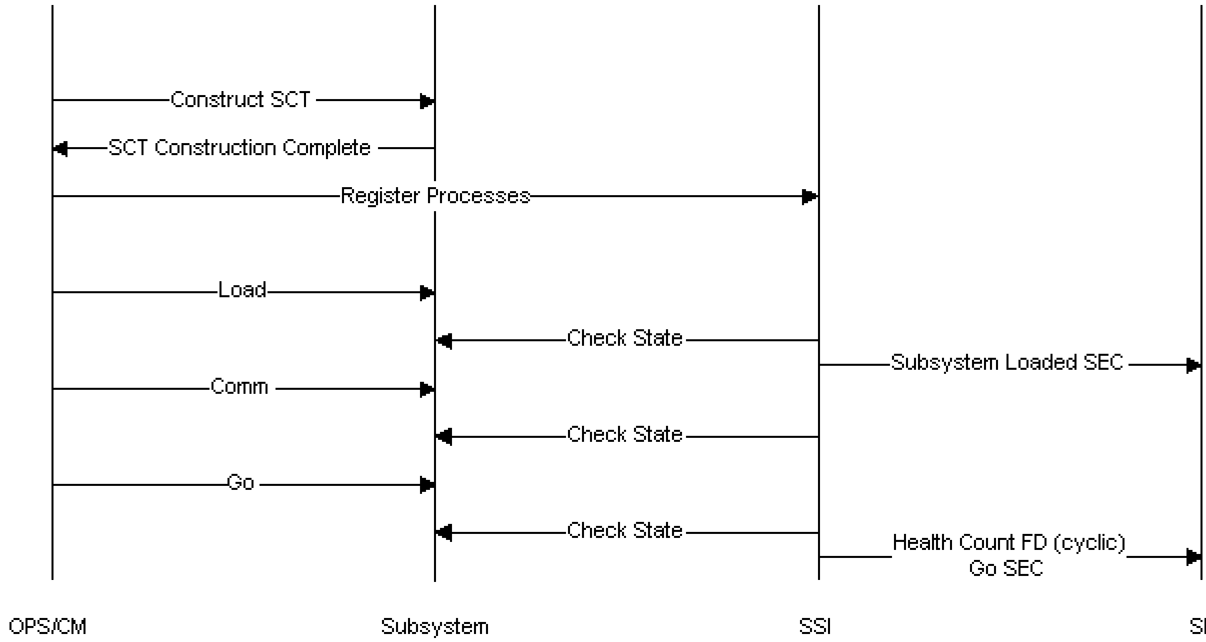


Figure 3. OPS/CM SSI Interactions

Figure 4 shows the typical interactions between an cyclic application process and Subsystem Integrity. Acyclic interactions between SSI and SI are identical to the cyclic operations with the exception that if the BeatEnd is specified, SSI checks for the PID rather than the heartbeat at its next scheduled health check. At an application-specified rate, the Application Process invokes the appropriate process heartbeat method. This updates the shared memory Heartbeat for the process. Periodically, at 10ms intervals (100ms for 10 Hz SSI), the SSI process checks for those heartbeats that should have arrived in the last cycle (not all of them). If all expected have been received, as well as having met additional health criteria as specified by SSI, it then increments the subsystem health counter and sends it to SI via the Health Counter FD. If at some point an expected heartbeat is not received from a process marked as essential in the SCT, SSI decrements the health counter instead of incrementing it and sends it on to SI. Any Process failure results in the generation of a Process Running FD with the appropriate state set.

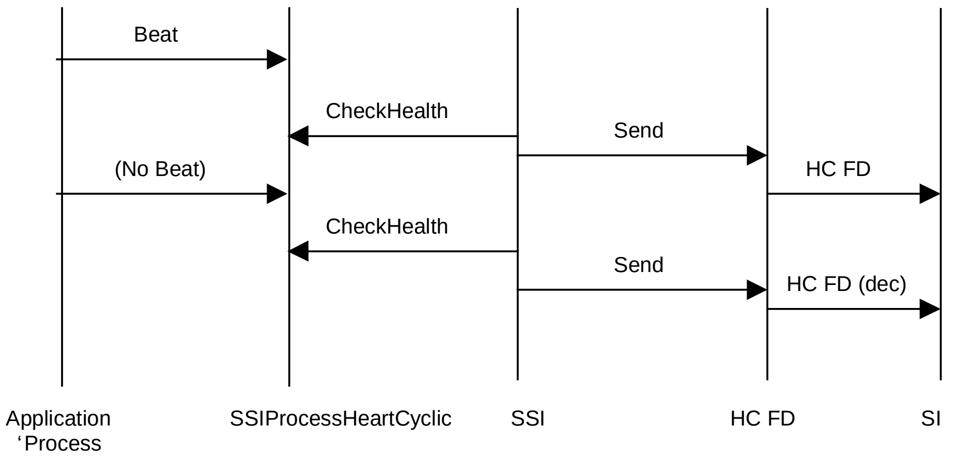


Figure 4. Application-SSI Interactions

1.1.3.2 Subsystem Integrity Context Diagram

SSI has limited visibility directly with the system. All APIs, with the exception of SSISubsystem, communicate through shared memory. SSI does communicate with FD Services (FDs), System Services (SECs), Syetem Message Services (Process Error Messages). Refer to Figure 5 for the system context of SSI.

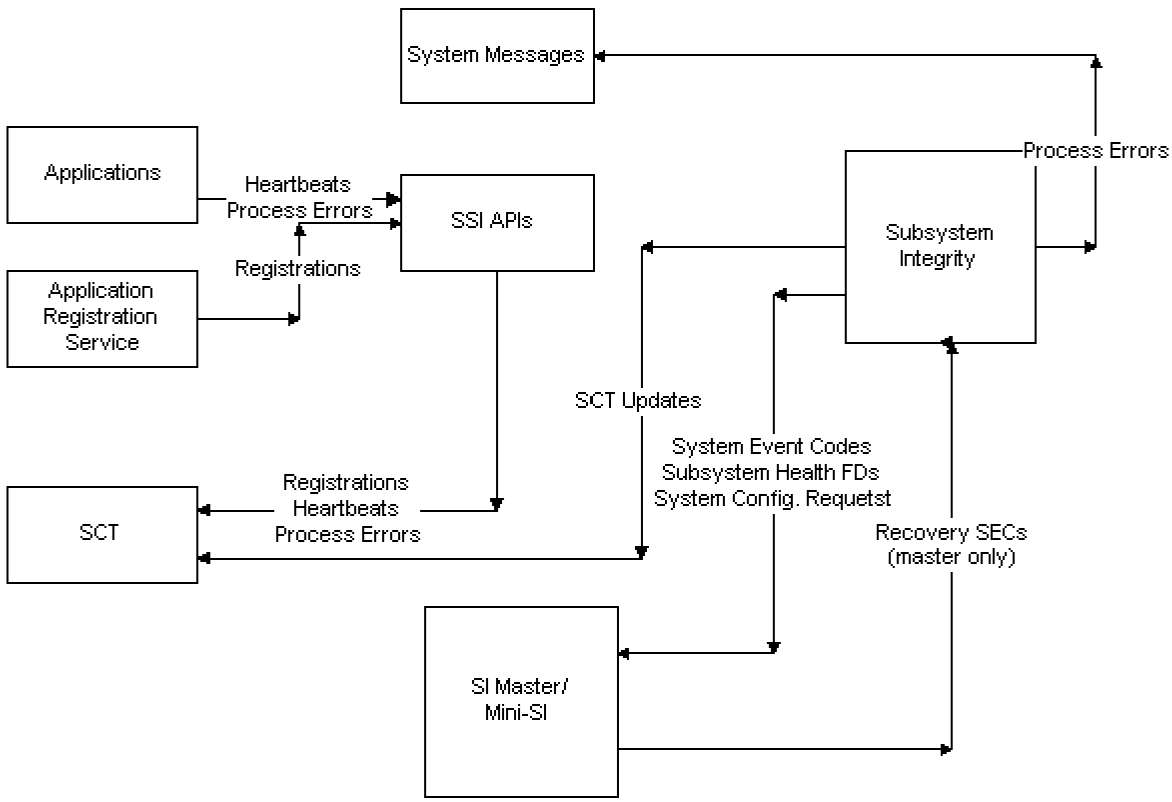


Figure 5 SSI Context Diagram

1.1.3.3 Subsystem Integrity State Definition and State Transition Diagram

One of the mechanisms by which SSI tracks the current health of a process is through the process states. As part of the regular subsystem health check at the DSR or SSR rate, subsystem cycles through each of the processes allocated to its process container, and queries each of its processes as to their health. In an effort to increase the efficiency of this poll and reduce the time required to exhaust the container, only registered processes are to be scanned. Once a process is registered, either formally or informally, as described in the registration and heartbeat API definitions in this document, it is then placed within the appropriate subsystem process container and its state is locally transitioned to a registered state. Once registered, the information is sent to the Master SCT, where it is published globally. All process state information is maintained globally in the SCT. When polled for its health, a registered process can be in one of several states. A description of the possible states and how they are determined is defined below.

- Not Registered: A process is by default in this state. No action occurs as a result of this state.
- Registered: A process is transitioned into this state when it is either formally or informally registered. A process running FD, with the registered state set, is sent out as well as a process name FD is issued with the corresponding process number as determined by the registration. SI sends out the FDs.
- Running: Once a process provides its first heartbeat or resumes a normal heartbeat, it transitions into this state. SI issues a process running FD, with the running state set.
- Stopped: Should a process fail to provide at least one heartbeat within its specified period and the process exists, it is transitioned into this state. SI issues a process running FD, with the stopped state set.
- Failed: If a process fails to provide a scheduled heartbeat, or is currently in the stopped state and ceases to exist, then it transitions into this state. SI issues a process running FD, with the failed state set.
- Complete: A process transitions into this state when the process is unregistered. SI issues a process running FD with the complete state set.

Once a process is unregistered, a process running FD, with the complete state set, is issued. The process is then removed from the process container of SSI and is therefore no longer polled for its health; potentially reducing the number of processes SSI must poll in order to determine its own health. The state diagram is shown by Figure 6.

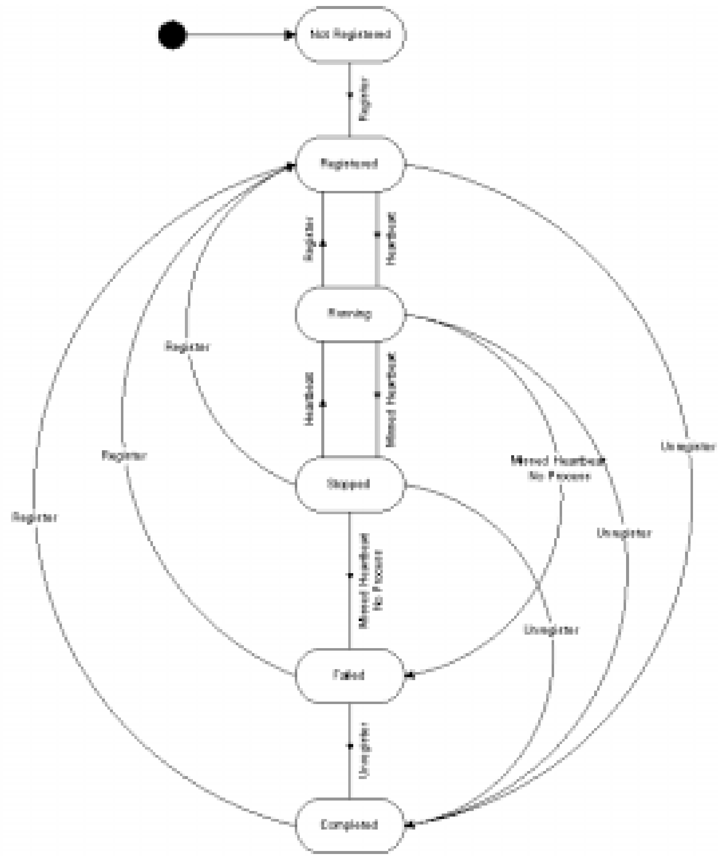


Figure 6 Process State Transition Diagram

A table driven polymorphic state machine is utilized to drive the states. This technique has been chosen for both efficiency and completeness. The approach ensures that all possible state combinations are represented guaranteeing that the system is able to properly handle any valid state change request as well as all possible invalid change requests. The state change classes are represented by Figure 7. In order to accomplish the global maintenance of the process states, two new system event codes are utilized, these are 442 (SSI to SI) and 443 (SI to SSI).

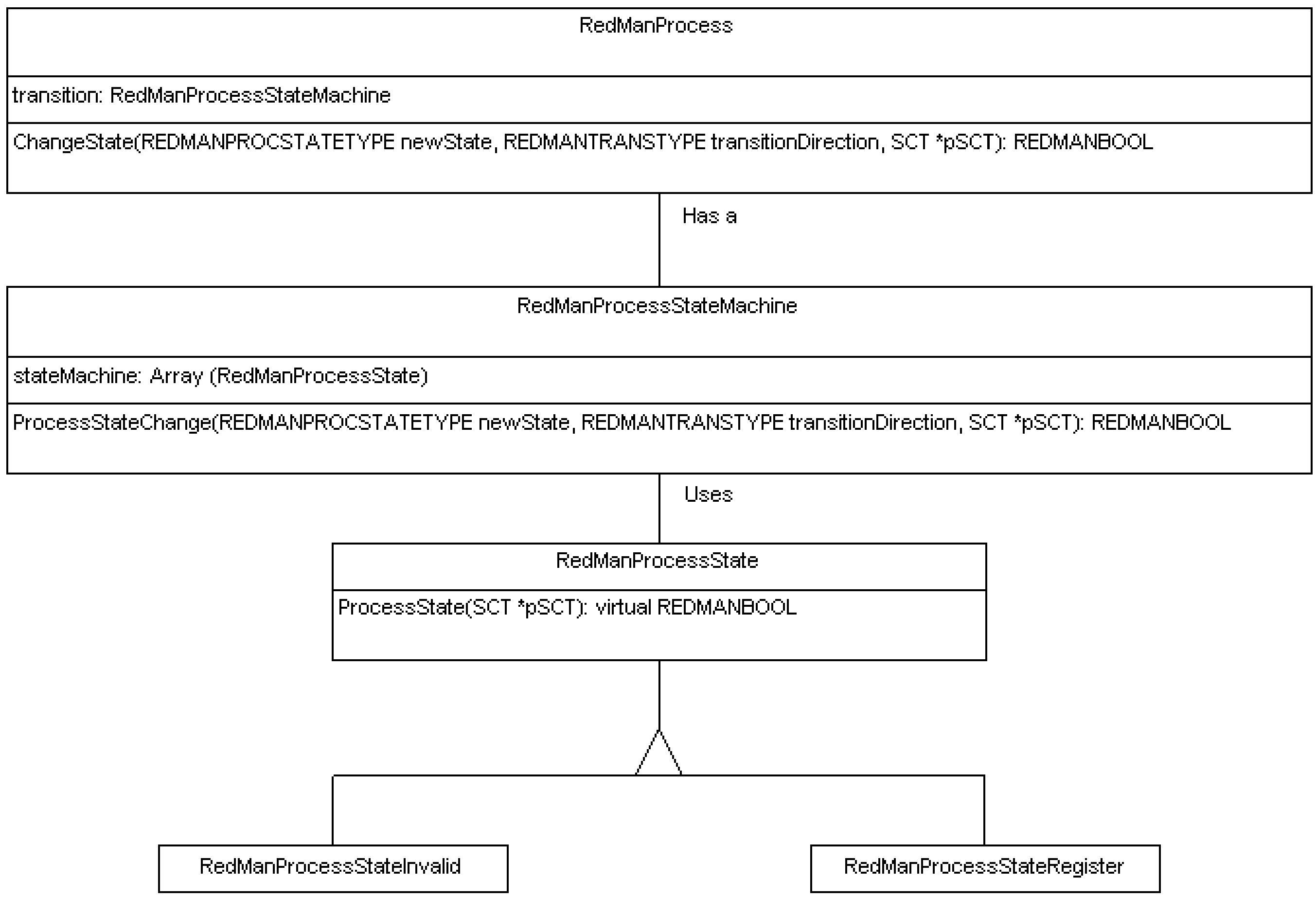


Figure 7 Process State Class Diagram

1.1.3.4 Subsystem Integrity Unique Algorithm Design

The process heartbeat algorithm within SSI is designed to work with both acyclic and cyclic heartbeats. Two modes of heartbeat operation will be provided to accomplish the monitoring capability, i.e., a heartbeat mode and a PID mode. Each process within SSI controls and monitors its own heartbeat data and reporting. The algorithm is presented in Figure 14.

1.1.3.5 Subsystem Integrity Development Tools

The tools outlined in the following table are used to implement Subsystem Integrity.

Table II Subsystem Integrity Development Tools

Tool(s)	Use
Misc. Text Editors	Code development
ProDev	UT and UIT
Visio Technical	Diagrams (Class, Sequence, DFD, etc.)
Word	Document preparation

1.1.3.6 Subsystem Integrity External Interfaces

The external interfaces are provided in the sections that follow.

1.1.3.7 Subsystem Integrity Data Dictionary

TBD

1.1.3.8 Subsystem Integrity Message Formats

Message Id: SCT_PROCESS_ERROR
Message Type: Details
Message Severity: Error

Process %s in subsystem %s failed. Reason Code: %d %s.

Insert 1: Process name
Insert 2: Subsystem name
Insert 3: User Defined Numeric reason code
Insert 4: User Defined Text reason

Help Text:
The process specified (insert 1) notified Subsystem Integrity of the Error specified (Insert 4).

1.1.3.9 Subsystem Integrity Display Formats

Subsystem Integrity has no displays.

1.1.3.10 Subsystem Integrity Input Formats

Subsystem Integrity accepts no external inputs.

1.1.3.11 Subsystem Integrity Recorded Data

All Subsystem to System Integrity data transfer is via either FDs or RM messages (SECs), and is therefore automatically recorded. SSI records no other data.

1.1.3.12 Subsystem Integrity Local Storage Requirements and Formats

Subsystem Integrity requires has no local tables or files.

1.1.3.13 Subsystem Integrity Printer Formats

Subsystem Integrity prints no data.

1.1.3.14 Subsystem Integrity Interprocess Communications (C-to-C Communications)

Shared memory is the primary vehicle for interprocess communications. Interfaces resident in non-Redundancy Management programs write all data to shared memory in local (to the computer) areas of the SCT. Subsystem Integrity executes periodically, at the SSR or DSR, and checks this data for changes. These changes are taken into account in the generation of the HC FD, which is also part of the SSR or DSR process. The C++ class definitions and the compiler define the format of this data.

There is no cross processor communication between parts of SSI.

1.1.3.15 Subsystem Integrity External Interface Calls (e.g., API Calling Formats)

1.1.3.15.1 API Layer Hierarchy

In an effort to provide the APIs while minimizing the indirect dependencies a user of these APIs may encounter, Redundancy Management (RedMan) has built several layers into its API architecture. Figure 8 represents these layers as the RTPS layer, the Apps layer and the RedMan layer, each of which are discussed in detail in the sections that follow. In addition, it also displays the accessibility of these layers to those users, depicted in the diagram as "Users", of the Redundancy Management services..

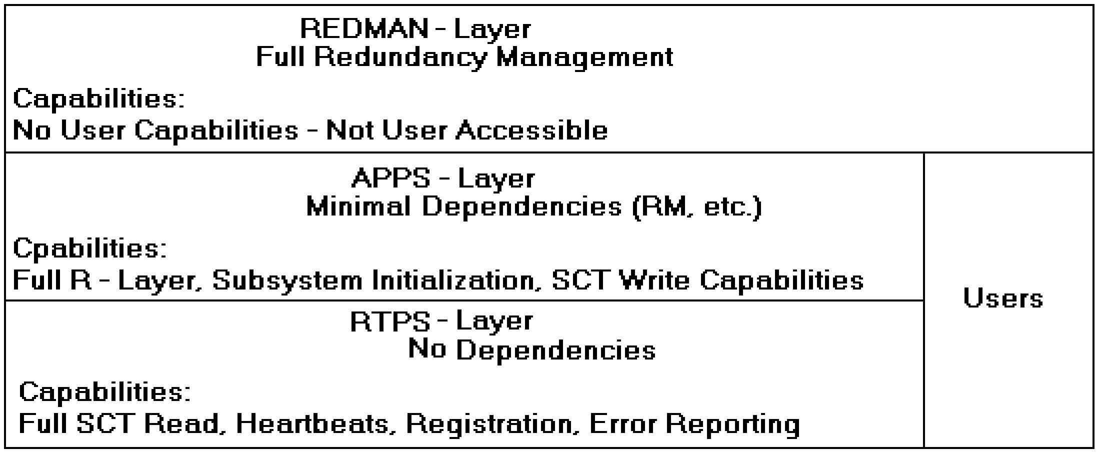


Figure 8 API Layer

1.1.3.15.1.1 RTPS Layer

The RTPS Layer, or R-Layer, is the simplest (lowest) of the layers. It is designed to provide full SCT read support, Process Heartbeats, Process Registration Services, and Process Error Reporting without linking in any additional services other than that being provided by the API. Thereby minimizing any unforeseen link dependencies, thus eliminating any circular link dependencies resulting from the use of the supplied API. In addition to minimizing the link dependencies, this layer is designed to minimize the resource and runtime impact on the user of the API. All communications necessary by Subsystem Integrity at this layer is accomplished through shared memory, utilizing semaphores where appropriate to ensure data integrity.

1.1.3.15.1.2 Applications (APPS) Layer

The Application Layer is the next layer within the hierarchy. At this layer it begins to bring in dependencies beyond the API, e.g., RM and its dependencies. This layer has been designed to provide authorized SCT writes, Subsystem Initialization, as well as providing full RTPS Layer capabilities. As with the RTPS Layer, this layer is constructed with minimal dependencies in mind.

1.1.3.15.1.3 Redundancy Management (RedMan) Layer

The RedMan Layer is the full implementation of Redundancy Management. This layer includes System Integrity, Subsystem Integrity, Computer Integrity, etc. This layer is not accessible outside of RedMan. It provides no user interfaces.

1.1.3.15.2 API Descriptions

Subsystem Integrity provides the following Application Programming Interfaces, each of which is presented in detail in the subsections that follow.

- Process Registration
- Process Heartbeats
- Process Error Reporting
- Subsystem Initialization

Descriptions of these APIs are available at:
http://www-clcs/project/syscontrol/redman/ClassList.html

1.1.3.15.3 Process Registration API

Through Subsystem Integrity, an API is provided allowing processes to be formally registered. Typically, processes are not directly registering themselves, that is a controlling process (ITS, CMP, HMP) formally registers those processes it has control over. All essential processes can only be registered through this interface. In order to accomplish the global maintenance of the registration, two new system event codes will be utilized, these are 446 (SSI to SI) and 447 (SI to SSI).

1.1.3.15.3.1 Process Registration Functional Description

Process registration is maintained globally in the SCT by registering the process with Subsystem Integrity locally, then Subsystem Integrity notifies the Master SCT of the registration through System Integrity. System Integrity notifies the Master SCT and it performs a registration of the process. Once registered in the Master, it then distributes the registered process notification out to all of the Client SCTs where they perform a similar registration, thus finalizing the propagating of the registration out to all SCTs. The unregistration of the process is performed in the same manner. The act of registering a process reserves a process object within the SCT corresponding to the process being registered. If a process was previously registered then unregistered, and is being registered again, then an attempt is made during the registration to provide the same SCT process object, thus preserving the process name and process number relationship from one registration to another. However this relationship is not guaranteed since SSI has a predetermined number of processes it can have registerer at any given instance, if SSI needs to register a process it will select an available process in the SCT for registration purposes; possibly a formerly unregistered process thus making the associated process number unavailable until the process is unregistered... In order to eliminate dependencies, this API has been implemented at the RTPS Layer. Therefore, all communication between the API and Subsystem Integrity is accomplished through shared memory. The notification between Subsystem Integrity and the Master SCT as well as the Master SCT and the Client SCTs is accomplished via System Event Codes. The process name and the process running FD are issued by the Master SCT once the Master receives the registration notification, thus eliminating any duplicate FDs from being generated. The process number under which the process was registered is provided as a result of the registration, as well as through an additional interface within the API. The sequence diagram in Figure 10 represents this description.

1.1.3.15.3.2 Process Registration Class Diagram

The implementation of the Process Registration requires several new classes be developed within this CSC. These classes include the API as well as new classes at the RTPS layer allowing RedMan the ability to perform the process registration without 1) having link dependencies, and 2) without exposing the underlying behavior required to accomplish the registration. The existing, as well as the new classes required, can be seen in the class diagram of Figure 9.

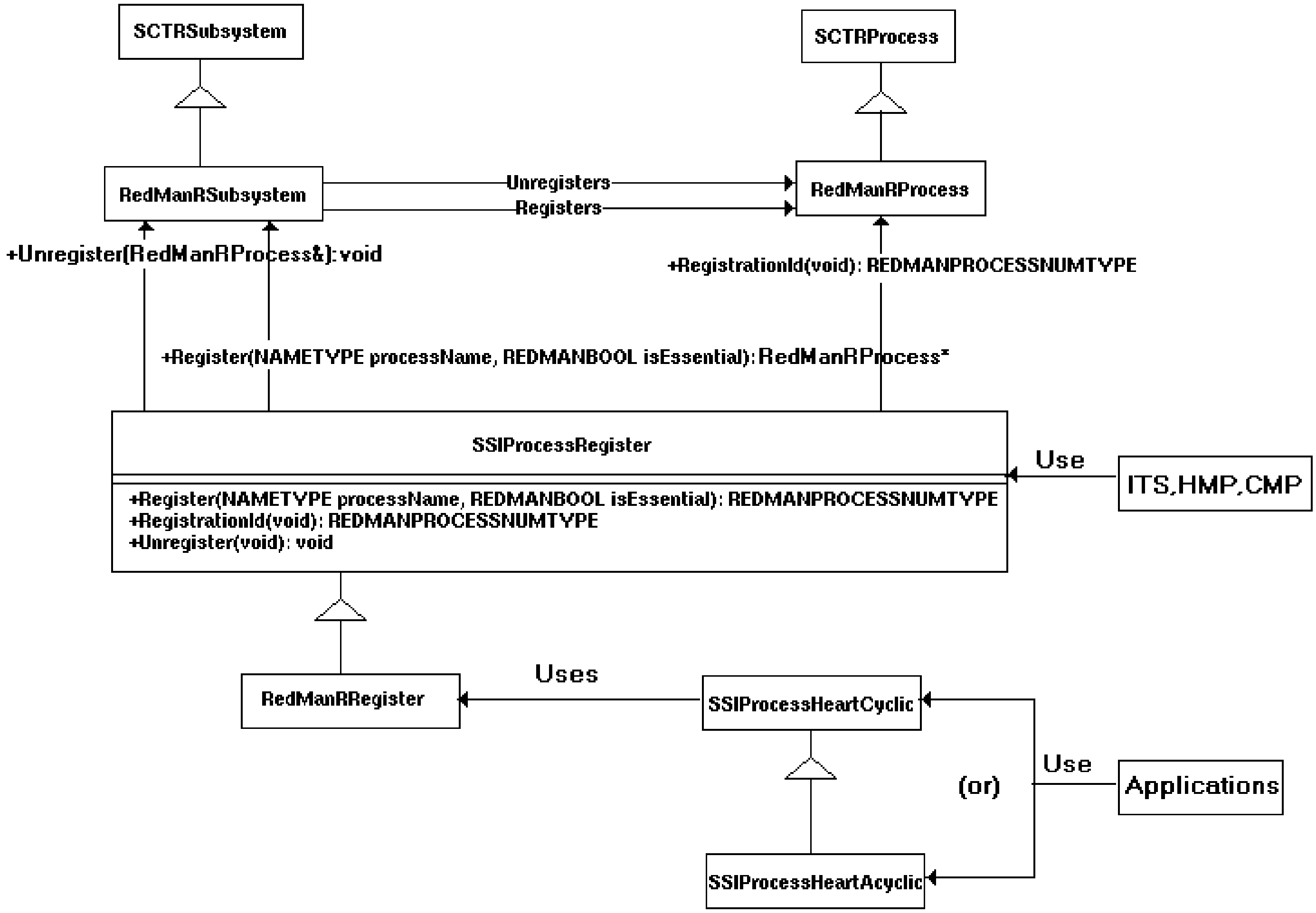


Figure 9 Process Register Class Diagram

1.1.3.15.3.3 Process Registration Sequence Diagram

The sequence diagram for the registration operations can be found in the figure below.

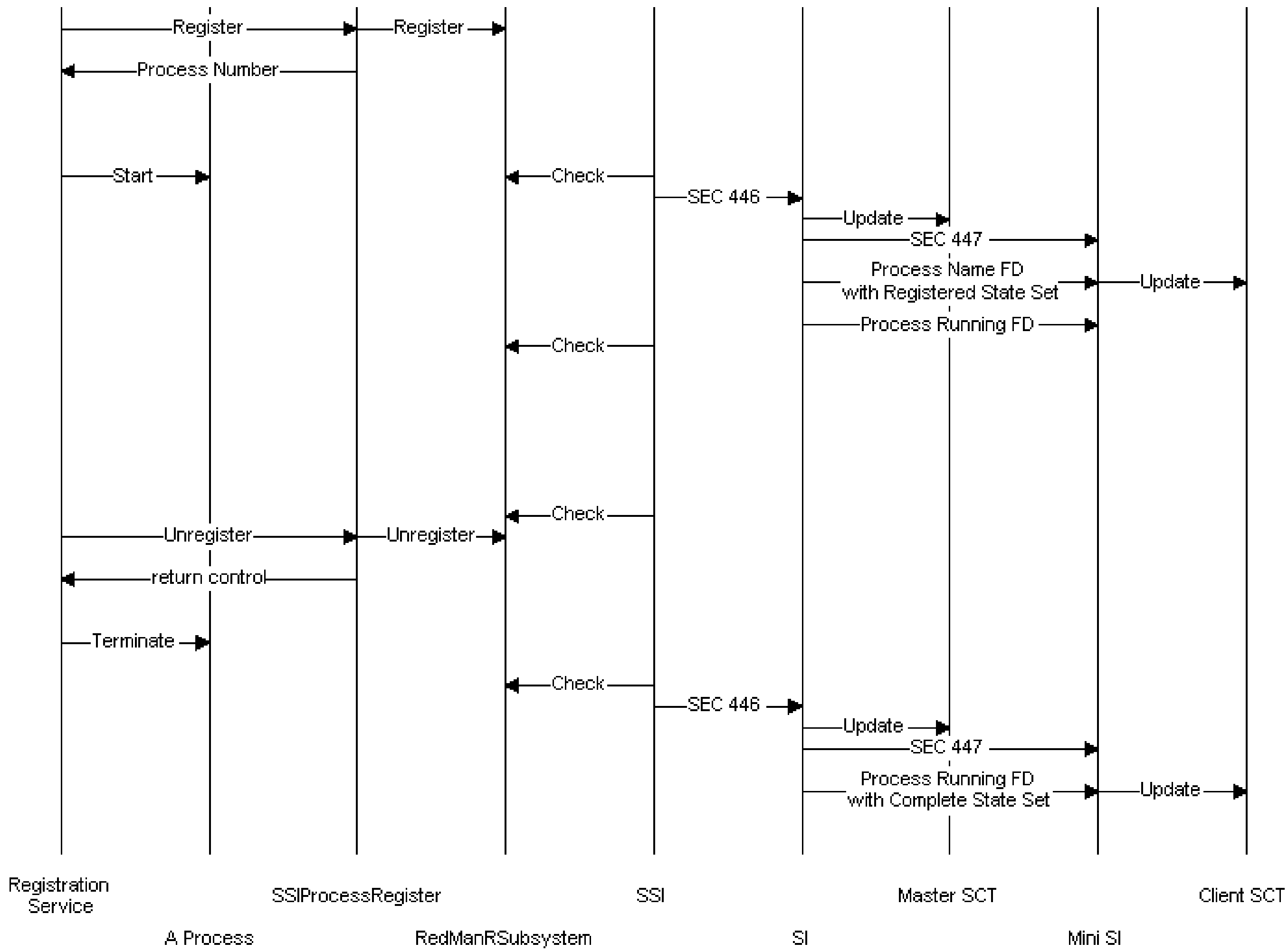


Figure 10 Process Registration Sequence Diagram

1.1.3.15.4 Process Heartbeat API

Through Subsystem Integrity (SSI), an API is provided allowing processes to periodically or aperiodically inform Subsystem Integrity that they are alive and processing. *All RTPS process must report a heartbeat to SSI.* In addition, all application processes must report a heartbeat to SSI. In an effort to minimize the impact on a process using the heartbeat API, it is implemented at the RTPS layer within the Redundancy Management API hierarchy whereby all communications to SSI is accomplished through shared memory.

1.1.3.15.4.1 Process Heartbeat Functional Description

There are two flavors of heartbeats provided by SSI. These are the Cyclic and the Acyclic heartbeats. At the moment a process constructs either of the heartbeat types, the heartbeat object attempts to register the process. One of two actions can occur. First, if the registration object determines that the process is currently registered, then it simply provides the process object to the heartbeat object and returns control. Second, if the registration object determines that the process is not currently registered, then the process becomes registered however the registration is an informal registration, i.e., the process is considered non-essential. Therefore it is imperative that the controlling process, that initially performed the formal registration, provide the registered process with the name it was registered under. If the process does not provide the heartbeat API with the name it was registered under, then a type two-(2) registration occurs causing the process to become registered as an informal process. If this happens, then the possibility exists that a process formally registered as an essential process has its heartbeats monitored as though it were a non-essential process. All informal registrations produce non-essential processes, unless formally registered as essential. Once the process has been registered, either formally or informally, then a process has been reserved within the SCT corresponding to that process allowing communication between the heartbeat API and SSI. Currently the number of possible processes represented in the SCT is 100, however this number is configurable. It is important to note that the Process Running and Process Name FDs need to exist in the OLDB equivalent or exceeding the number of processes supported by SSI for each subsystem within the TCID. Once the heartbeat API has been constructed, the matter is simply to provide a heartbeat to SSI. This is done in one of two ways, either cyclically or acyclically.

- Cyclic Heartbeat: The cyclic heartbeat is provided by the process at the rate provided to the heartbeat object during construction. To accomplish this, a Beat() interface has been provided. Once the process provides its first Beat(), then this is an indication to SSI to start watching the process for heartbeats. From this point, SSI continually monitors the process until the heartbeat object is destroyed. From the first Beat(), until the object is destroyed, it is the responsibility of the process to provide heartbeats to SSI at the specified rate. It is okay to provide more than one heartbeat within the specified period, however if during the specified period the process does not provide at least one heartbeat, then the process is considered to be unhealthy by SSI.
- Acyclic Heartbeat: The acyclic heartbeat is constructed with a default period, provided by the process utilizing the API. To accomplish the acyclic heartbeat, a process has the ability to start the heartbeat monitor using the default period provided in the constructor, or specify a period for that heartbeat. Once the BeatStart() interface is utilized then SSI begins to monitor the process for a beat within the period specified. During that beat period, the process has the ability to extend the beat for a particular period by issuing a BeatExtend(). Once received, it indicates to SSI that it should recalculate the current heartbeat rate for that cycle. The end of the beat cycle is completed by the process sending a BeatEnd()through the API. This informs SSI to discontinue its monitoring of heartbeats for this process until the next BeatStart() occurs. This interface provides the ability of truly event driven processes to provide a heartbeat to SSI only when necessary, i.e., when they need to process an event. As with the cyclic heartbeat, it is okay for the process to complete its heartbeat cycle before the specified period, however if it should fail to complete the cycle within the designated rate, then the process is considered to be unhealthy by SSI.

It is important to note that it is possible for an acyclic process to die following the issuance of a BeatEnd()indicating to SSI to stop watching the heartbeats from a process. This could provide a false indication to SSI that the process is simply not processing any events. In an effort to eliminate this false indication, the acyclic process monitoring has two modes of operation, 1) during a valid heartbeat cycle, SSI monitors for heartbeats 2) once the heartbeat cycle completes, SSI begins to monitor the PID for its existence in the PID Table within the OS. The PID is extracted from the OS and stored within the SCT once the heartbeat API is constructed by the process. An error messaeg is produced if a process has not been formally registered at the time of the informal registration attempt. The sequence diagrams in both Figure 12 and Figure 13 represent this description.

1.1.3.15.4.2 Process Heartbeat Class Diagram(s)

The implementation of the heartbeat classes is designed in order to take advantage of the API layer architecture displayed in Figure 8. In order to adhere to the layer restriction while providing both the cyclic and the acyclic interfaces, several new classes are introduced. All of the classes as well as their interactions are shown in Figure 11.

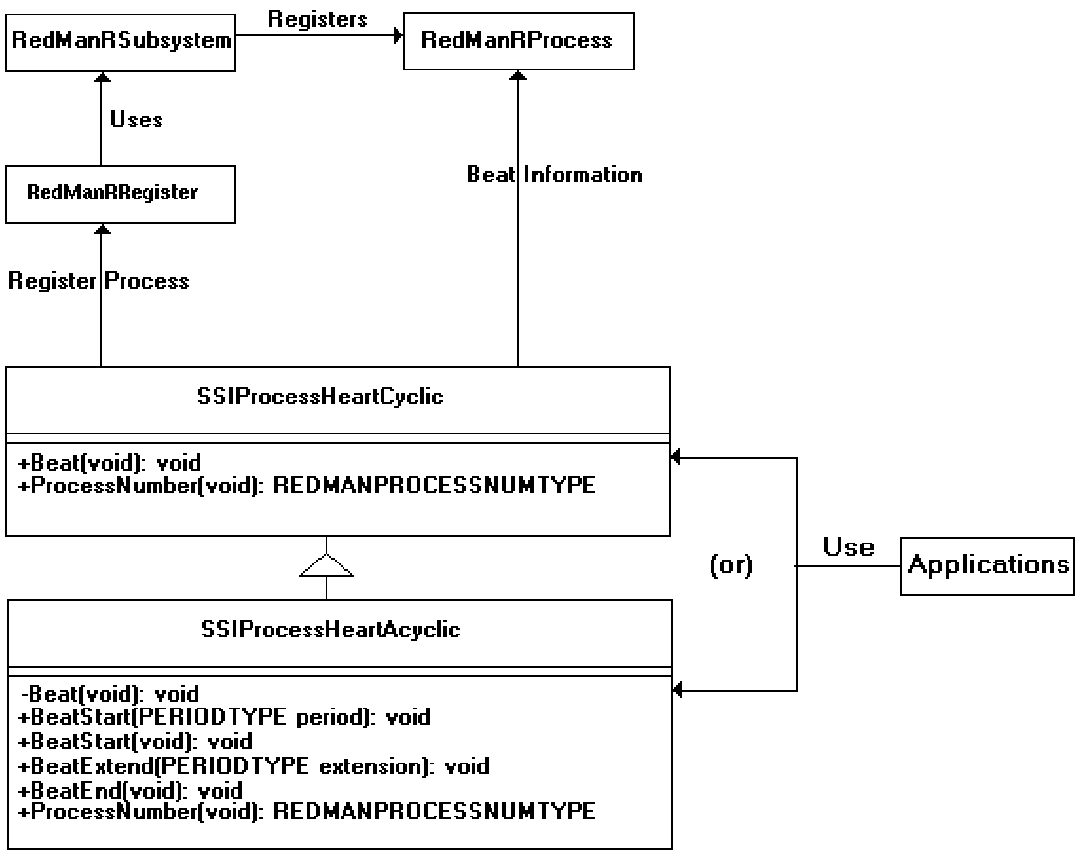


Figure 11 Process Heartbeat Class Diagram

1.1.3.15.4.3 Process Heartbeat Sequence Diagram(s)

The sequence diagrams for both the cyclic and the acyclic heartbeat operations can be found in the figures below.

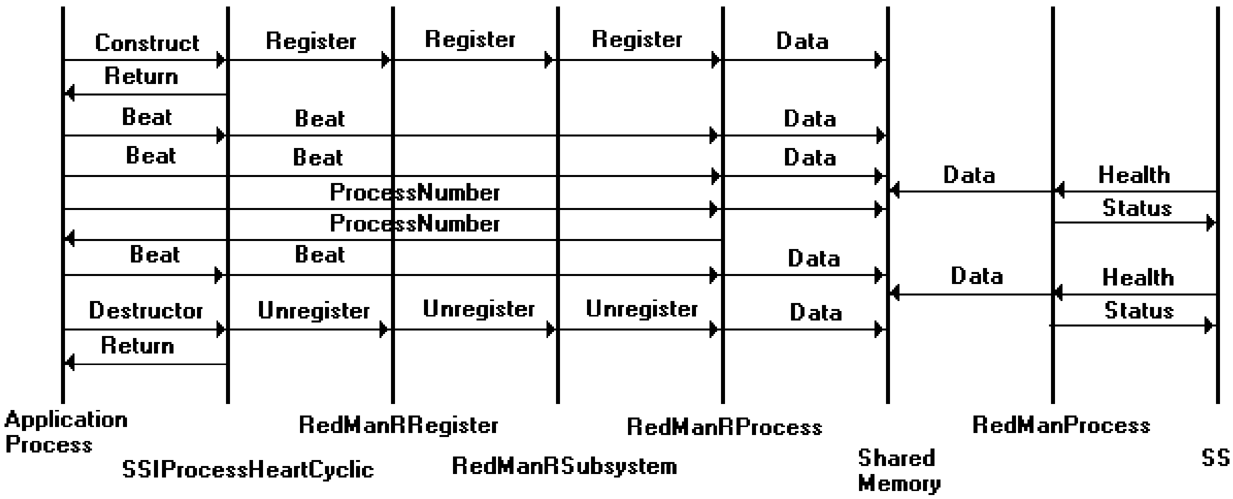


Figure 12 Cyclic Heartbeat Sequence Diagram

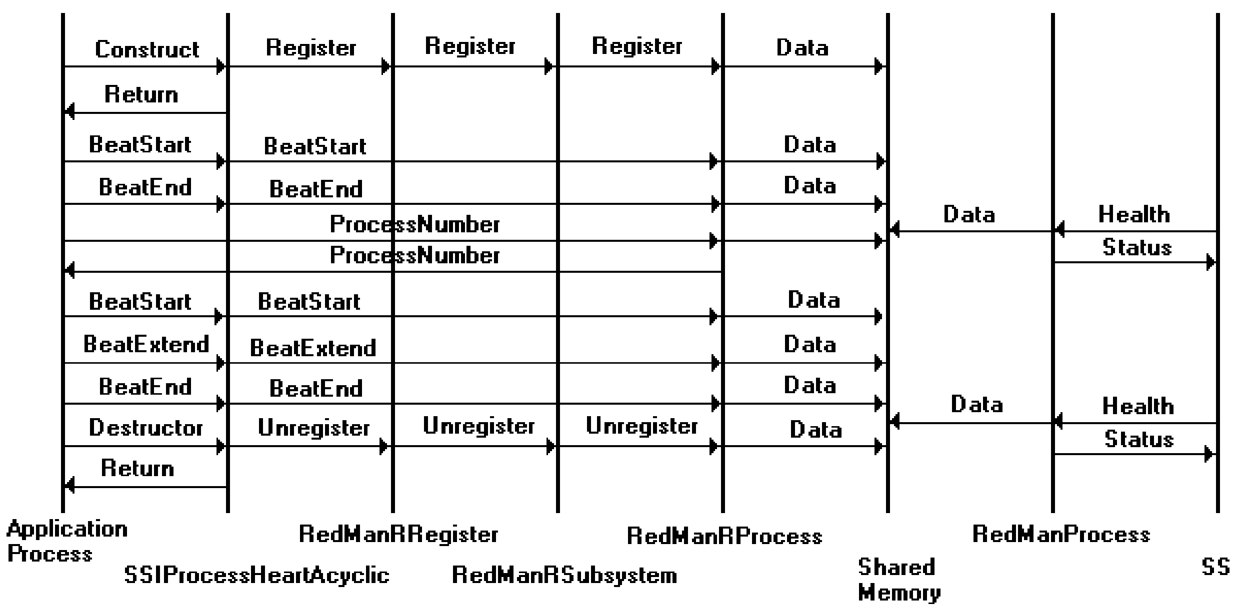


Figure 13 Acyclic Heartbeat Sequence Diagram

1.1.3.15.4.4 Process Heartbeat Algorithm

Figure 14 represents the heartbeat verifications algorithm utilized by SSI when on its scheduled health check.

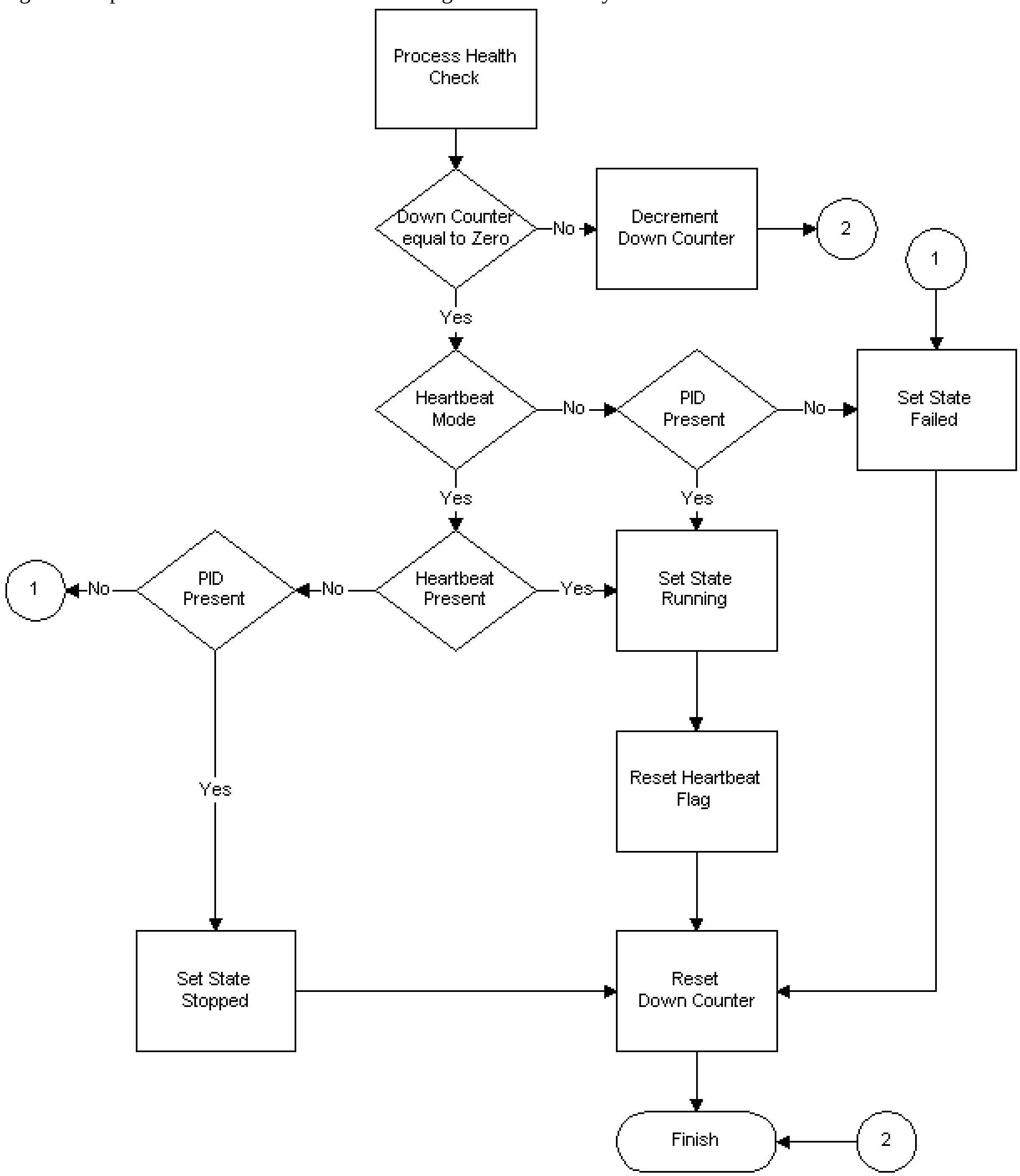


Figure 14 SSI Heartbeat Algorithm

1.1.3.15.5 Process Error Reporting API

Through SSI, an API is provided allowing processes the ability to report runtime errors to SSI. These errors are one of the mechanisms that SSI utilizes in determining the health status of a subsystem. An error has two attributes, 1) the error type, and 2) the error severity. A description of each is presented below.

- Error Type:
  - Data: Typically this type of error indicates that a process has encountered an error with its data, i.e., a data element does not contain an expected value.
  - Execution: Typically this type of error indicates that a process has encountered an error in its logic. The software may have encountered a section of source code that it was not expecting to enter during normal operations.
  - Interface: Typically this type of error indicates that a process has encountered an error with a data element passed through its interface. It may be that the data is corrupted or empty (null pointer/value) causing some type of default processing to occur.
- Error Severity:
  - Information: Typically this level of error indicates that a process has encountered a minor error; it (the process encountering the error) is able to recover completely from the error and continue unaffected; the error condition simply needs to be logged.
  - Warning: Typically this level of error indicates that a process has encountered an error and recovery is attempted by the process encountering the error, if necessary or possible. This may indicate that the normal operations of the process, from the point at which the error was reported may be at risk; possibly affecting the overall operability of the process.
  - Error: Typically this level of error indicates that a process has encountered a severe error in its processing and it (the process encountering the error) will attempt to recover if possible. This error may as well indicate that system operability is at risk.
  - Fatal: Typically this level of error indicates that a process has encountered a severe error in its processing and it (the process encountering the error) is unable to recover. Processing from the point where the error is encountered may be futile. This error indicates that overall system operability is at risk. Once an error of this nature is encountered and the process is essential, SSI decrements its health counter and SI transitions the affected subsystem to a communicating state. The affected process remains in the failed state until it is either unregistered *or cleared by SSI.*

1.1.3.15.5.1 Process Error Reporting Functional Description

The process error reporting is implemented at the lowest API layer to eliminate link dependencies. Therefore all communications necessary to report the error to SSI occur through shared memory. Once a process constructs an instance of the error reporting API, the new error object locates the local subsystem (SSI) thus creating a direct communication path between the error object and SSI through shared memory. When the process is encountered with an error and sends a Report() message to the error object, the message is delivered directly to SSI through the API. Based upon which network the subsystem is located on, either the DCN or the RTCN, the reported error messages will be flushed by SSI to System Messages at a DSR or SSR respectively. The sequence diagram in Figure 16 represents this description.

1.1.3.15.5.2 Process Error Reporting Class Diagram(s)

The implementation of the process error reporting API requires modifications to the current error reporting class in order to accommodate the RedMan API layer architecture represented by Figure 8. The new class is depicted below as Figure 15.

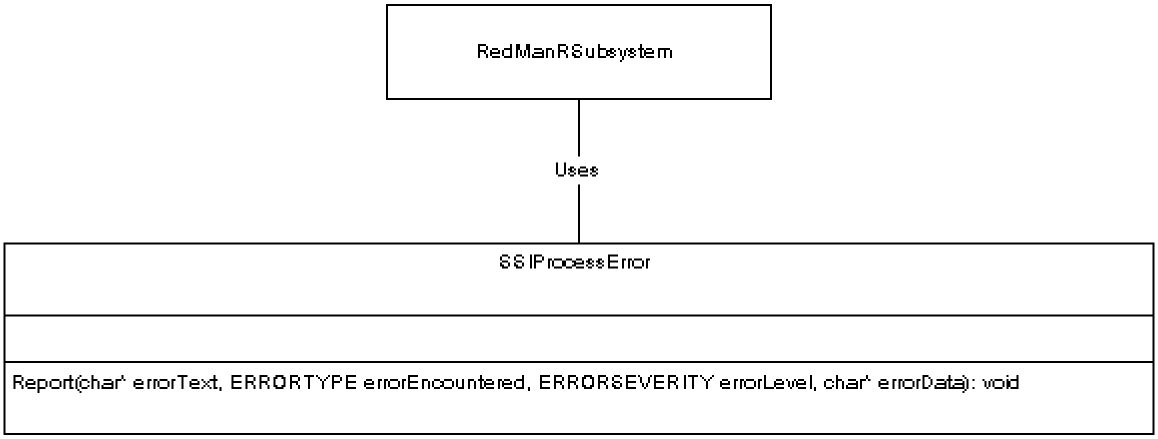


Figure 15 Process Error Class Diagram

1.1.3.15.5.3 Process Error Reporting Sequence Diagram(s)

The process error sequence diagram is shown below.

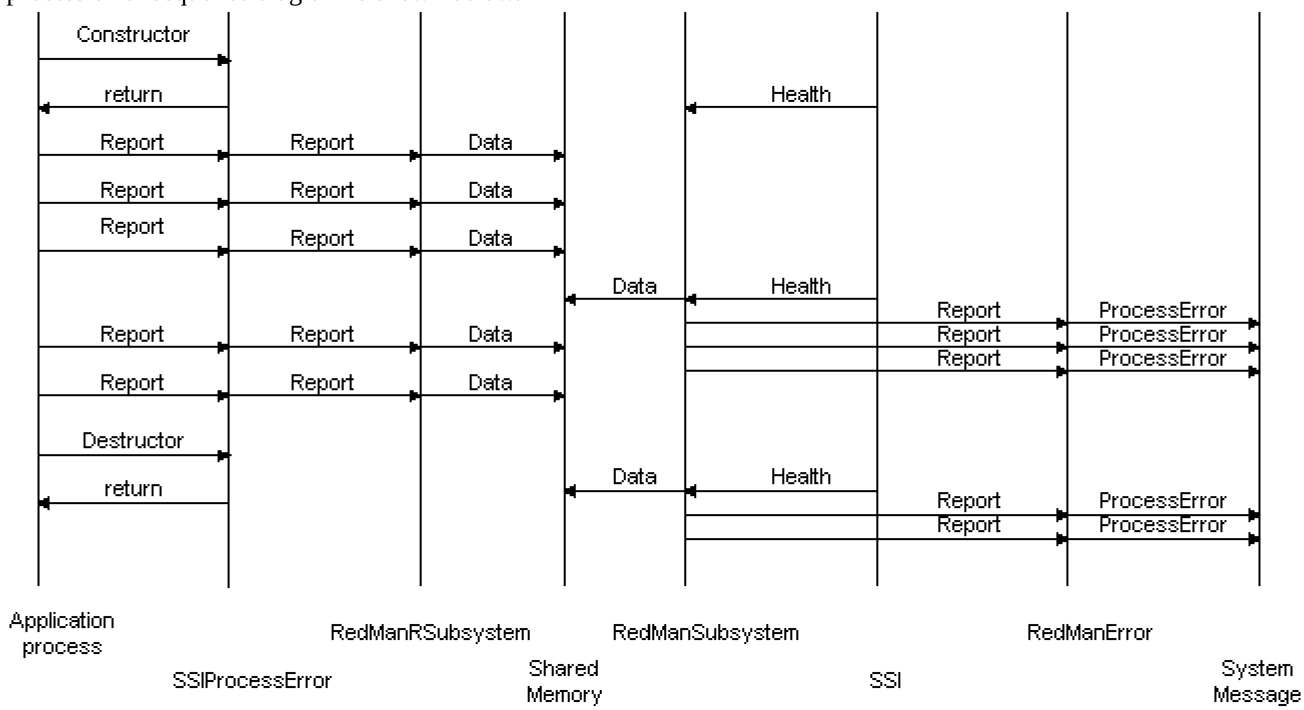


Figure 16 Process Error Sequence Diagram

1.1.3.15.6 Subsystem Initialization API

The SSISubsystem provides an interface for Ops/CM to notify SI that the box has completed the transitions to Loaded and Go. (Other transitions are determined by data available to SI). Thor 3.0 does not affect this interface.

1.1.3.15.6.1 Specification:

class SSISubsystem

```
{
public:
SSISubsystem();
void     Loaded();
void     Go();
void     NoGo();
void     ~SSISubsystem ();
}
```

1.1.3.15.6.2 SSISubsystem

This constructor creates the shared memory connection so that the OPS/CM process can communicate this information with the SI SSR process. This constructor must be invoked after SI has been loaded.

SSISubsystem();

Arguments: N/A
Return Value: N/A

1.1.3.15.6.3 Loaded

This interface is to be invoked when OPS CM has successfully loaded the subsystem. Invocation results in the Loaded SEC being generated and sent to SI.

void Loaded();

Arguments: N/A
Return Value: N/A

1.1.3.15.6.4 Go

This interface is to be invoked when OPS CM has successfully started all System Software. Invocation results in the Go SEC being generated and sent to SI.

void Go();

Arguments: N/A
Return Value: N/A

1.1.3.15.6.5 NoGo

This interface is to be invoked when OPS CM succeeds in loading Subsystem Integrity but fails loading other system software. Invocation results in the No Go SEC being generated and sent to SI.

void NoGo();

Arguments: N/A
Return Value: N/A

1.1.3.16 Subsystem Integrity Table Formats

Subsystem Integrity uses no tables.

1.1.4 Computer Integrity

Computer Integrity's role is to gather performance data and determine whether a platform is healthy. Although multiple subsystems may be present on a single platform (CCP/DDP), only one copy of Computer Integrity will execute on that platform.

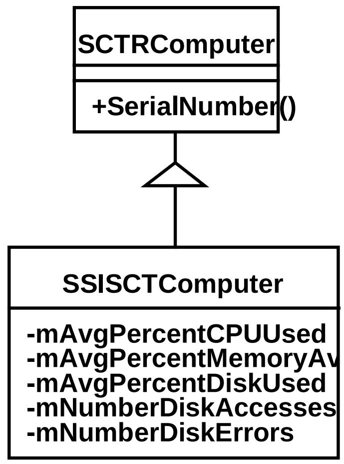


Figure 17. Computer Integrity Classes

Computer Integrity is composed of the following class:

SSISCTComputer: This class gathers the performance data from the OS and determines whether the platform is healthy. The private data attributes of the SSISCTComputer Class among others are inputs to a method that determines Computer health.

1.1.4.1 Computer Integrity Detailed Data Flow

Computer Integrity gathers performance data from the operating system (OS) at 10 second intervals, publishes the gathered data as SECs and FDs (see Requirements), and determines if the platform is healthy.

Computer Integrity also monitors Unix kernel errors through the syslogd message services (see Table 1.1.1.1). For Thor, if an error is received from the OS kernel with a priority of LOG_CRIT or below, the platform will be marked as unhealthy. Obtaining error messages about the Unix kernel through syslogd is not entirely satisfactory for a number of reasons:

- Locating a reason for the error would involve scanning the message text for keywords **-** No standard format - Highly OS specific
- It is undocumented what specific platform conditions would differentiate LOG_EMERG vs. LOG_ALERT vs. LOG_CRIT

.
Unfortunately, syslogd seems to be the only semi-portable method for obtaining information about hardware errors from the OS. Other solutions are currently being sought.

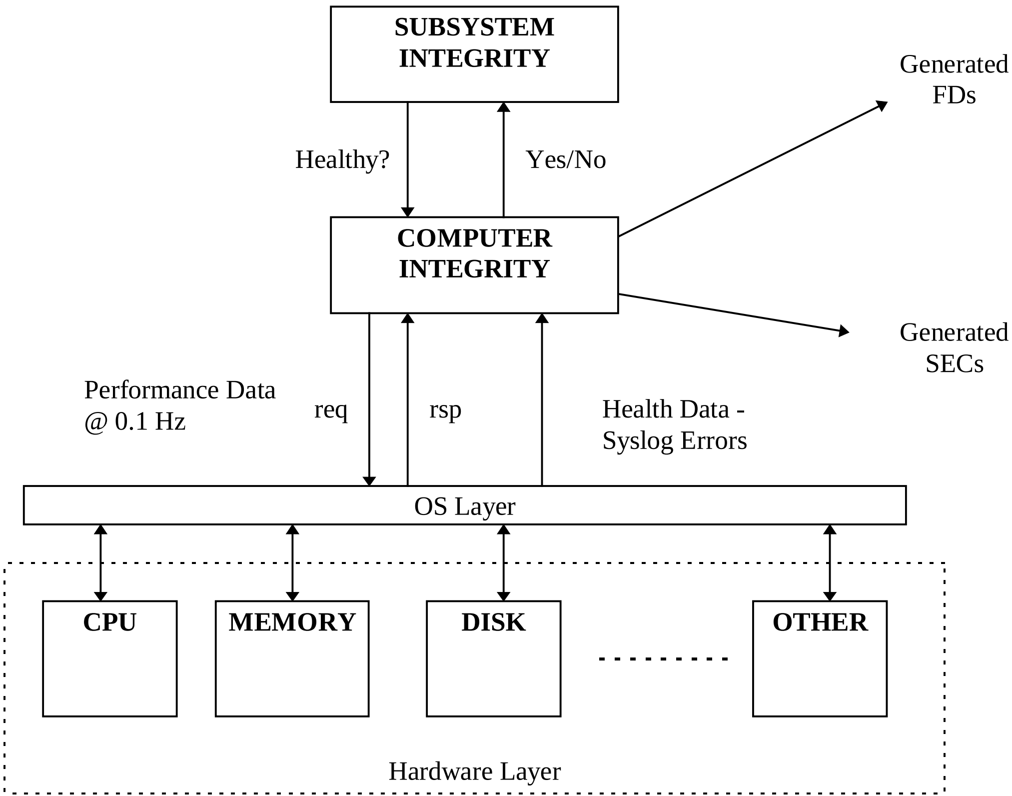


Figure 18. Computer Integrity Data Flow

Table III. SYSLOG Message Priority Types

Message Priority Type	Description	Unhealthy if recvd?
LOG_EMERG	A panic condition. This is normally broadcast to all users.	Yes
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database	Yes
LOG_CRIT	Critical conditions, e.g., hard device errors	Yes
LOG_ERR	Errors	No/*Maybe*
LOG_WARNING	Warning messages	No/*Maybe*
LOG_NOTICE	Conditions that are not error conditions, but should possibly be handled specially.	No/*Maybe*
LOG_INFO	Informational messages	No
LOG_DEBUG	Messages that contain information normally of use only when debugging a program	No

1.1.4.2 Computer Integrity Context Diagram

TBD

1.1.4.3 Computer Integrity State Definition and State Transition Diagram

TBD

1.1.4.4 Computer Integrity Unique Algorithm Design
TBD

1.1.4.5 Computer Integrity Development Tools
TBD

1.1.4.6 Computer Integrity External Interfaces
TBD

1.1.4.7 Computer Integrity Data Dictionary
TBD

1.1.4.8 Computer Integrity Message Formats
TBD

1.1.4.9 Computer Integrity Display Formats
Computer Integrity owns no displays

1.1.4.10 Computer Integrity Input Formats
Computer Integrity does not have a language-like interface

1.1.4.11 Computer Integrity Recorded Data
TBD

1.1.4.12 Computer Integrity Local Storage Requirements and Formats
TBD

1.1.4.13 Computer Integrity Printer Formats
Computer Integrity prints no data

1.1.4.14 Computer Integrity Interprocess Communications (C-to-C Communications)
Computer Integrity hardware performance data and configuration data (serial number) are stored in the local copy of the SCT . Computer Integrity also generates FD and SEC packets of performance data as shown in Table IV and Table V.

Table IV. FDs Generated by Computer Integrity

FD NAME	DESCRIPTION	STYP	LEN

FD NAME	DESCRIPTION	STYP	LEN
SnnnnnDISKU	CCWS001 DISK USE COUNTER	DEC	16
SnnnnnDISKE	CCWS001 DISK ERROR COUNTER	DEC	16

Table V. SECs Generated by Computer Integrity

SEC Number	Name	Source	Destination
408	CPU Utilization	CI	SI (Master SCT)
409	Available Memory	CI	SI (Master SCT)
410	Disk Utilization	CI	SI (Master SCT)
411	Disk Accesses (DISK USE COUNTER)	CI	SI (Master SCT)
412	Disk Errors (DISK ERROR COUNTER)	CI	SI (Master SCT)

1.1.4.15 Computer Integrity External Interface Calls (e.g., API Calling Formats)

Computer Integrity provides no API calls

1.1.4.16 Computer Integrity Table Formats

Computer Integrity uses no tables.

1.1.5 System Configuration Table

After system initialization, the master copy of the System Configuration Table resides on the Master CCP. Synchronized copies are maintained on all other computers via System Event Codes (SECs). Each SEC contains an 8-byte code which encapsulates such information as the sender's logical address, the sequence number, and the event code.

On each computer, the SCT is initially built from pre-stored files, *then updated based on the changes that have been recorded at the Master CCP*. The SCT is maintained in shared memory, and an API is provided to allow the SCT users to get access to the information. The system configuration can be viewed as a tree – the Set contains Test Sets, each Test Set is a group of Subsystems, each Subsystem is a collection of processes. The structure of the SCT, and the design of the API reflect this organization. Iterators are provided to allow looping through the sets to extract information, shortcut "My" objects are also defined at each level to allow quick access to local configuration data.

Initial creation of the SCT is through Microsoft Access tables. These tables allow much of the configuration to be defined prior to the start of the test, then updates as the system is configured.

1.1.5.1 System Configuration Table Detailed Data Flow

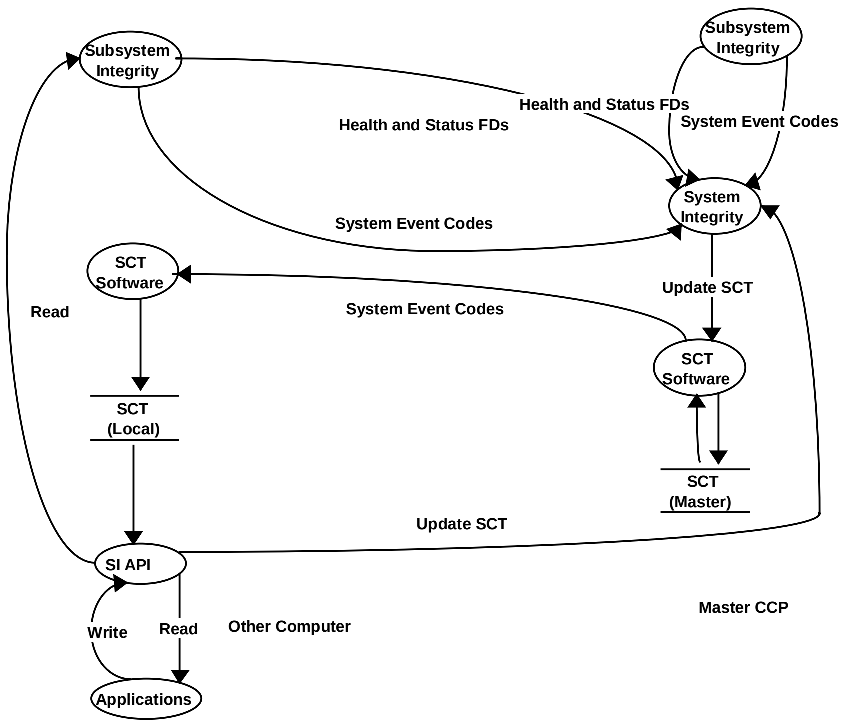


Figure 19. System Configuration Table Data Flow

When an application at a computer requests a modification to the SCT through the provided API, the update request is sent to the master copy of the SCT located on the Master CCP. There, the update is made as appropriate, and the other SCTs are updated through System Event Codes. The updated data is then available to the application through the Read APIs. Subsystem Integrity updates are handled in much the same way. Subsystem Integrity reports health and Status through both System Event Codes and Health and Status FDs. Based on this input, System Integrity evaluates the health of the subsystem and updates the master copy of the SCT. These updates are relayed to all copies of the SCT through System Event Codes.

1.1.5.2 System Configuration Table Context Diagram

TBD

1.1.5.3 System Configuration Table State Definition and State Transition Diagram

TBD

1.1.5.4 System Configuration Table Unique Algorithm Design

TBD

1.1.5.5 System Configuration Table Development Tools

TBD

1.1.5.6 System Configuration Table External Interfaces

Since the SCT is brought up early within a Subsystem initialization routine, System Messaging will not be available until a later time frame within the initialization sequence and any error reporting accomplished by the SCT will be to a local temporary data store provided by the SCT. However, once the necessary mechanisms are in place allowing the SCT to utilize System Messaging, the following System Messages may be generated by this CSC.

Message Number = SCT_ILLEGAL_TRANSITION
Message Type = Details
Severity = Error

Illegal state transition occurred: Subsystem %s from %s to %s.

Insert #1= Character String (subsystem name)
Insert #2 = Character String (old state)
Insert #3 = Character String (new state)

Help Text:
This message indicates that a state transition occurred in the named subsystem that was not expected based on the design. Possible causes include: Incorrect System Integrity Design, Unforeseen initialization sequences, data corruption.

This event will also cause the SCT to be logged to the SDC.

Message Number = SCT_SUBSYSTEM_STATE_CHANGE
Message Group = Details
Severity = Informational

Subsystem %s state changed from %s to %s.

Insert #1= Character String (subsystem name)
Insert #2 = Character String (old state)
Insert #3 = Character String (new state)

Help Text:
The named subsystem changed state. This message indicates the state change occurred as a planned transition under normal operations.

Message Number = SCT_UNEXPECTED_SUBSYSTEM_STATE_CHANGE
Message Group = Summary
Severity = Error

Subsystem %s Unexpectedly changed state from %s to %s

Insert #1= Character String (subsystem name)
Insert #2 = Character String (old state)
Insert #3 = Character String (new state)

Help Text:
This message indicates that a failure occurred in the named subsystem. That subsystem changed state. System Integrity may be taking recovery actions.

Additional Messages are TBD in the Detailed Design to indicate initialization errors during loading of SCT.

1.1.5.7 System Configuration Table Data Dictionary

TBD

1.1.5.8 System Configuration Table Message Formats

TBD

1.1.5.9 System Configuration Table Display Formats

This CSC does not provide Table Display Formats.

1.1.5.10 System Configuration Table Input Formats

Input formats for the Subsystem Integrity CSC are defined by the SEC Formats and the API.

1.1.5.11 System Configuration Table Recorded Data

The SCT will attempt to utilize RM for most of its messaging needs since the recording mechanism for the SDC is inherent within Reliable Messaging (RM). However, since the SCT is needed by RM and the SCT is early within the Subsystem initialization routine, it must utilize services lower within the Message Stack (see Notes) such as Application Messaging (AM) until which time RM is made available (post SCT build) to the SCT. The SCT will employ and restrict itself to the various packet types provided within the Thor RTPS Packet Payload ICD 84K00351-001 for recordability.

1.1.5.12 System Configuration Table Local Storage Requirements and Formats

TBD

1.1.5.13 System Configuration Table Printer Formats

This CSC does not print anything, therefore does not provide Printer Formats.

1.1.5.14 System Configuration Table Interprocess Communications (C-to-C Communications)

Because the SCT resides on all computers, and must be kept current on all computers, there are a number of internal operations involved in synchronizing and re-synchronizing the SCTs. These operations are described below.

1.1.5.14.1 System Configurtation Table Initialization

Typically, the Master CCP will be the first machine powered on, and will, as a result, also own the Master SCT, built from the files on its local disk. However, there are cases where the configuration needs to be changed prior to powering on the Master CCP. This can be accomplished by allowing the Master SCT to reside in places other than the Master CCP, then transferring ownership to the Master CCP when it is up and running. The following protocol is used to ensure that exactly one Master exists prior to creation of the Master CCP (See the state transition diagram below):

1) In the Initialization State, the SCT process issues a broadcast Request SCT for the current SCT using Application Messaging (AM). Reliable Messaging requires the SCT for its own initialization, and therefore cannot be used to load the initial SCT. One of three things will occur: The request will timeout, one or more Master Acknowledgments may be received, or a Master SCT response may be received.

2) If the Master SCT response is received, the process loads the SCT from the message, and at completion of the loading is the in the SCT Loaded State. This will be the typical path for computers other than the Master CCP when the Master CCP is brought up first.
3) If one or more Master Acknowledgments are received, the ownership of SCT Master is still being decided, but this computer is not an immediate candidate. It sits in the Waiting state while ownership is decided. Once decided, the owner will send the Master SCT Response, at which point the process loads the SCT from the response and continues to SCT Loaded state. There are some conditions under which the eventual master will not be aware that the process is waiting, so after a specified time-out, the Waiting process returns to the Initialization State where it repeats is broadcast Request SCT. Typically, the master will respond with the SCT, but occasionally, it will return to the wait state, and in even rarer cases (all competing masters have failed prior to any of them becoming master), it will compete to become the master. This state will be most frequently used when power is applied to all computers simultaneously.
4) If the Request SCT times out, the process enters the competition for ownership of the Master SCT. It then announces to the world its host name and its goal to become Master SCT via an Assert Master Broadcast. It repeats this announcement 3 times, waiting Init SCT Timeout between each announcement. Very few computers will reach this state. At least two annunciations are required. The second computer to reach the competing state will have missed the first assert. The first computer then needs to Assert a second time to notify the second computer of its rank. The third assert is only necessary if one of the messages is dropped. The first computer powered on will reach the Compete state; typically this will be either the Set Master CCWS or the Master CCP. If all computers are powered on simultaneously, one or two should reach this state, the rest should reach the Waiting state. The length of the timeout has some bearing on how many reach this state. The longer the timeout, the greater the number of competitors.
5) While in the Competing for Master state, the process may receive an Assert Master Broadcast from another computer. If it does, it uses ranks hostname against the hostname of its competitor. Earlier in alphabetical order is defined to be higher in rank. If it is of higher rank than the received broadcast, it ignores the broadcast. If of lower rank, it returns to the Init state and issues a new Request SCT. The higher ranking competing master will then respond with a Master Acknowledgment.
6) If no higher ranking computer Asserts during this period, this computer becomes the Master and completes loading the SCT from files. In addition, once entering the competition, it responds to any Request SCT broadcast with a Master Acknowledgment. This limits the competition to those computers that came up within the timeout window of the first Request SCT. Note that if two or more computers timed out on the Request SCT, future requesters may receive a Master Acknowledgment from more than one computer. A missed Request SCT may result in another computer trying to assert after this computer has reached the Loading From File state. At that point, the process responds with an Assert Master of that outranks all contenders.
7) After reaching the Master Loaded state, it responds to each earlier Request SCT (which it earlier responded to with a Master Acknowledgment) with a Master SCT Response. Future Request SCTs receive the Master SCT response that contains the current SCT.
8) Note that no computer can reach the SCT loaded state until the competition for master has been successfully completed. Because this is true, no state change SECs have been sent. All computers powered on during the competition have SCTs that reflect the state of the files on the computer that became master.

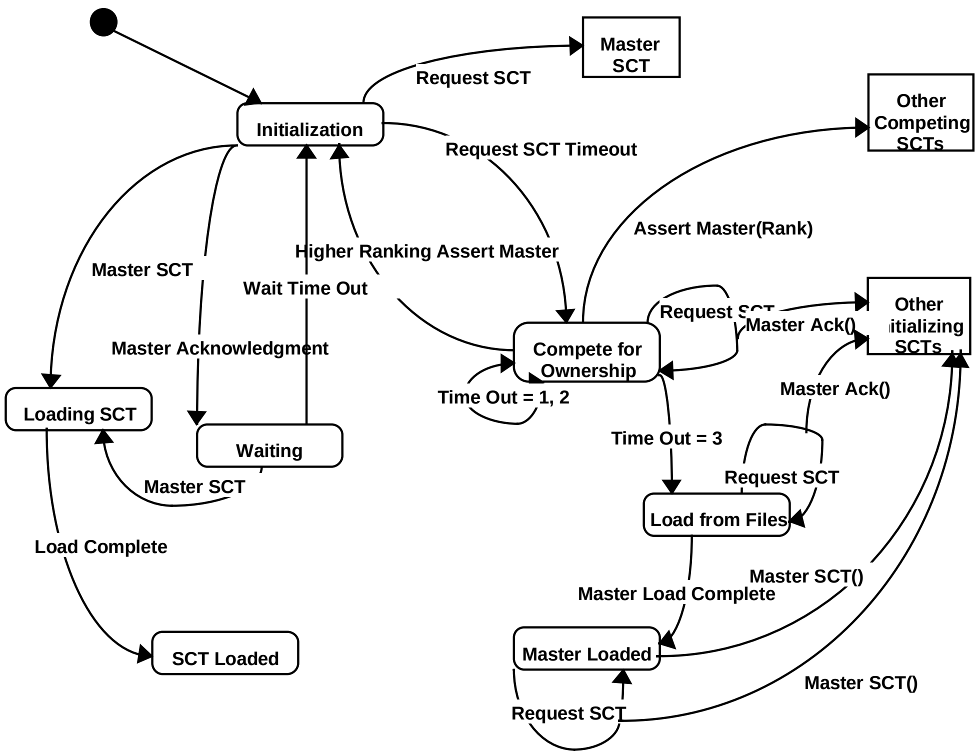


Figure 20. SCT Initialization State Transitions

1.1.5.14.2 System Configuration Table Transfer of Ownership

When the Master CCP comes up, it needs to assume ownership of the Master SCT. However, in order to allow for the Master CCP to be assigned, it cannot assume ownership based purely on file content. Assuming the Master CCP does not own the Master SCT when it completes its SCT initialization, it then initiates a process to claim ownership (See the state transition diagram below):

1) Prior to the Master CCP coming up, the Master SCT will be residing elsewhere within the Test Set (See SCT Initialization). Once the Master CCP loads its SCT per the SCT Initialization state machine, its SCT will be in an SCT Loaded state. From the SCT Loaded state, one of two things will occur, first if the SCT has any updates to process following the initial table load from the Acting Master SCT, it will process these updates, second the SCT will make its CCP Master host determination causing a transition from the SCT Loaded state to the Acquire Master state.
2) Immediately following the transition of the Master CCP SCT to the Acquire Master state, it will 1) begin storing all of the SCT update requests for historical reference, 2) send out a Request Master Relinquish message using Application Messaging (AM) to the Master SCT for it to begin the transfer process thereby causing the Master SCT to switch to its Relinquish Master state, and 3) begin assuming some of the responsibilities of the Master SCT by sending out Master Acks for any SCT request and storing them for later transmission. However, if a timeout occurs while in the Acquire Master state, the Master CCP SCT will simply re-issue the Request Master Relinquish message to the Master SCT.
3) While the Master SCT is still in the Master Loaded state, it accomplishes a couple tasks. First it processes any SCT update requests it receives from the System Integrity Stub as well as processing any SCT requests from other Initializing SCTs, second the Master SCT processes the Request Master Relinquish message from the Master CCP SCT by immediately storing the SCT updates, without processing, and transitioning into its Relinquish Master state.

4) Once the Master SCT is in the Relinquish Master state, it will continue to store any and all SCT update requests for historical purposes to be utilized by the CCP Master SCT during the transition process. The Master SCT will send a Master Ack for all SCT requests by other Initializing SCTs, and it will send out a Relinquish Response message with the stored SCT updates, including their sequence numbers, back to the Master CCP SCT as a confirmation of beginning the SCT transfer process. While in this state, if the Master SCT does not receive the Relinquish Accept message after sending out the Relinquish Response message using AM (up to 3 times), the Master SCT will timeout and change back to the Master Loaded state and process all outstanding SCT updates and requests.
5) It is important to note that at this time, when the Master CCP SCT is in the Acquire Master state and the Master SCT is in the Relinquish Master state, the Master CCP SCT will synchronize itself with the Master SCT by verifying that it has received and processed all of the SCT updates by comparing its last SCT update sequence number with that of the first one received within the Relinquish Response message from the Master SCT. If any of the SCT updates were not received by the Master CCP SCT, it will request them from the Master SCT and process, re-requesting if necessary. Thereby synchronizing the Master CCP SCT with the Master SCT.
6) Following the receipt of the Relinquish Response message from the Master SCT, the Master CCP will synchronize its collection of SCT update requests with those packaged in the Relinquish Response message from the Master SCT, to determine stale updates and any overlapping updates between the two collections. Once synchronized, it will send out a Relinquish Accept message using AM to the Master SCT and transition to the Master Loaded state.
7) While the Master SCT is in the Relinquish Master state and receives the Relinquish Accept message from the Master CCP SCT, it will flush its collection of SCT updates, requests, and discontinue sending out the Master Acks to the other Initializing SCTs. The Master SCT will then transition to the SCT Loaded state, completing the transfer of the Master SCT to the Master CCP SCT.
8) Once the Master CCP SCT has transitioned into the Master Loaded state, it will process all of the SCT updates and requests it has stored, new update requests, and new SCT requests, thus completing the SCT Master transfer.

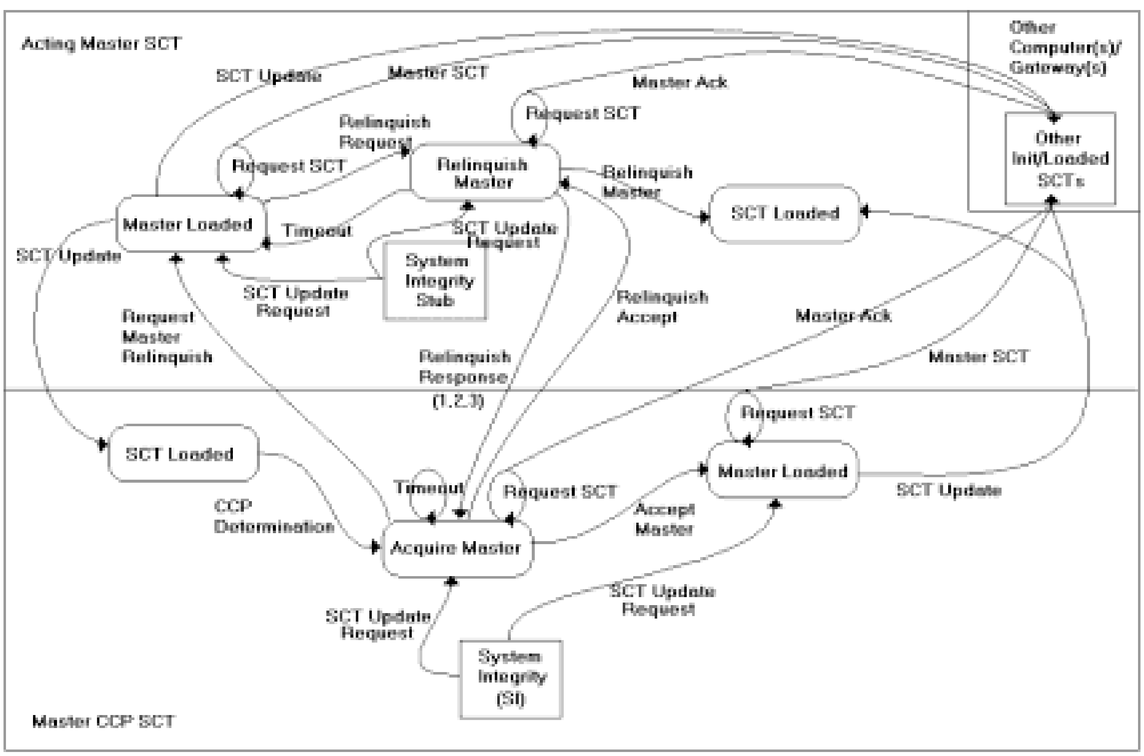


Figure 21. SCT Transfer of Ownership State Transitions

1.1.5.14.3 System Configuration Table Synchronization

All updates to the SCT are initiated by System Integrity at the Master CCP. The Master SCT, which resides on the Master CCP, then issues a SEC to all other copies of the SCT. This SEC contains the requested update, and a sequence number that specifies the current version of SCT. The Master SCT also publishes this version number as a System Status FD. When the SEC is received by a remote SCT, the version number is checked against the current version number to ensure no updates have been missed (SSI also verifies the current SCT version matches the FD as part of its SSR/DSR health check. See the Subsystem Integrity section, 1.3.3, of this document for details of the periodic check.). If the SEC sequence number is correct, the update is applied to the SCT. If not, the SCT requests all updates since its latest version directly from the Master SCT through a C-to-C. These updates are then applied as received.

Relevant System Event Codes:
256: Subsystem Loaded
257: Subsystem Communicating
258: Subsystem Go
259: Subsystem No Go
260: Subsystem Not Communicating
261: Subsystem Not Loaded
264: New Active
393: Subsystem is running ORT
394: Subsystem is not running ORT
401: Subsystem Loaded
402: Subsystem Communicating
403: Subsystem Go
404: Subsystem No Go
405: Subsystem Not Communicating
406: Subsystem Not Loaded
413: Initial HC Received
500: SCT Relinquish Request
501: SCT Relinquish Response
502: SCT Relinquish Accept

SECs from Master SCT to Slave SCTs for change requests are TBD in the Detailed Design.

1.1.5.14.4 System Configuration Table Shared Memory

An SCT will reside on each platform (HCI, CCP, DDP, Gateway) with all updates being received by the Master SCT. The individual SCTs residing on a Resource, including the Master SCT, will be referenced through its external interface provided by this CSC. Each SCT will create an area of shared memory where it will store the necessary objects comprising the SCT. The first object stored within this address space will be the Set object as depicted below.

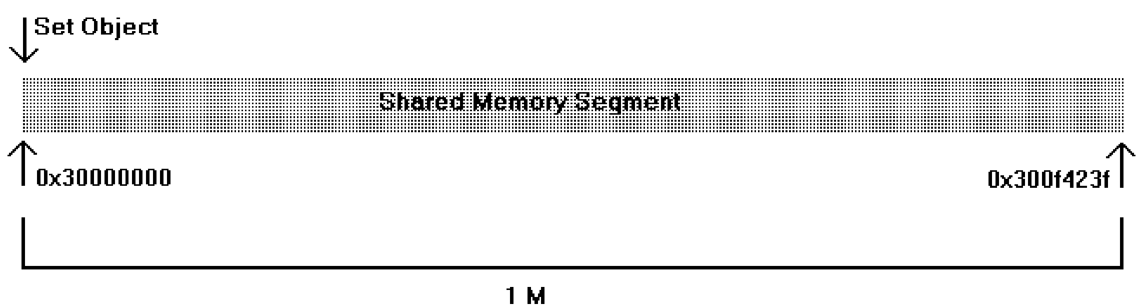


Figure 22. SCT Shared Memory

The SCT shared memory segment will be approximately 1 Megabyte in size. The final size of the memory space for Thor will be based upon the number of elements within the SCT initialization files, the current size is more than adequate for the SDE-1 data set, additional data sets will be made available for future testing. Each SCT API object constructed will attach itself to the appropriate region of shared memory and establish the necessary links into those objects containing the relevant information for the object constructed, i.e., if an SCTRSet, or SCTRTestSet object is constructed, links to its name, relevant “My” information, and appropriate containers are established.

1.1.5.15 System Configuration Table External Interface Calls (e.g., API Calling Formats)

This CSC provides an interface for reading data form the SCT as well as writing information into the SCT. The interface is comprised of multiple objects. As shown in Figure 23, the SCT is a tree of containers that reflects the logical configuration of the system. The Set is a collection of Test Sets, Each Test Set contains a number of Subsystems, Resources, Gateways and Groups. Resources are attached to external systems, and Subsystems are composed of Processes.

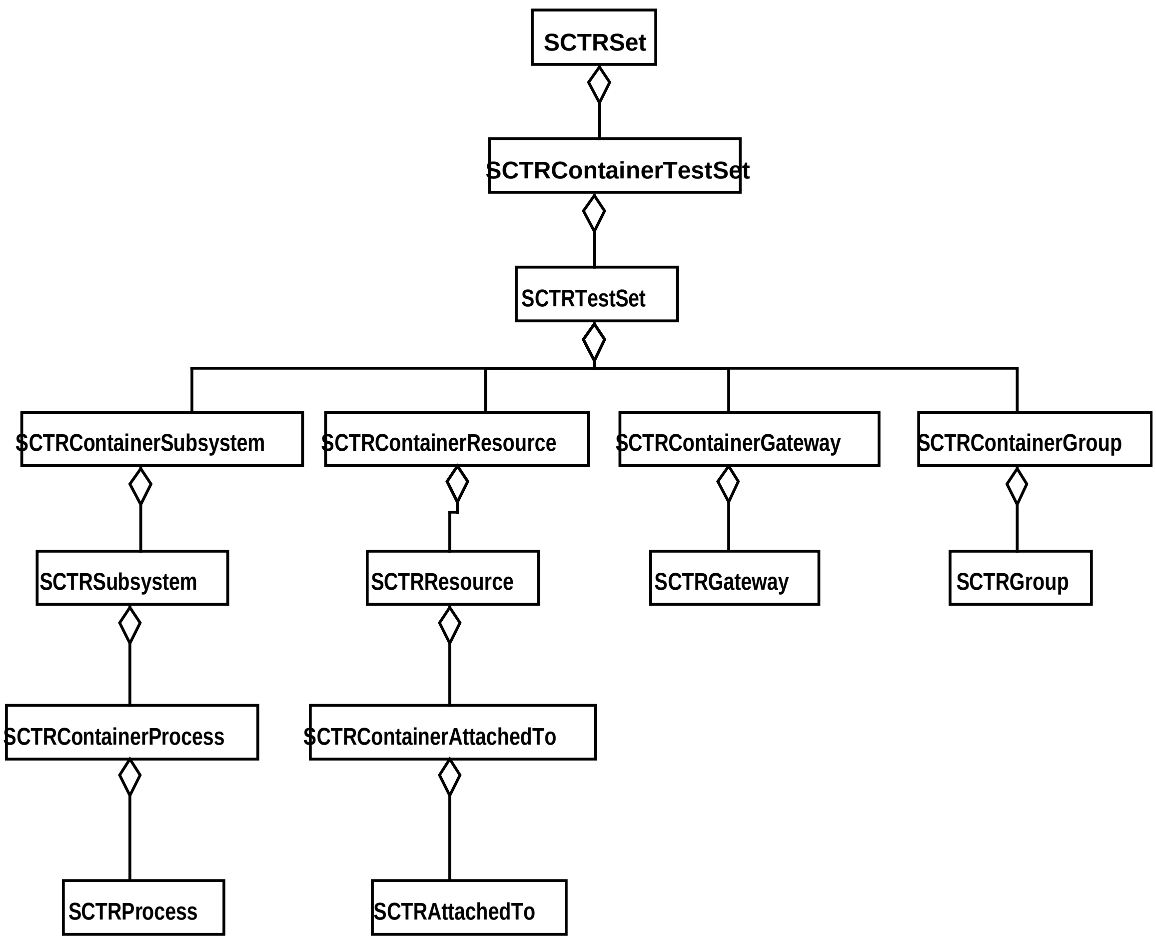


Figure 23. SCT API Hierarchy

The collections are all derived from the SCTOrdered class (Figure 24). While there is no inherent ordering of the various items, the use of the Ordered class allows searching for objects and stepping through each of the Objects. Only those methods that manipulate the existing members of the set are provided through the SCT Ordered class. This prohibits deletion of SCT elements through the API. No unique methods are provided in the SCT Ordered subclasses, but the methods are redefined to allow only homogeneous sets.

Figure 25 specifies the methods of each of the classes. In order to be collected, each is derived from the SCTCollectable class. All values in the classes are available as methods that return the values. The GetContainer methods in each class allow retrieval of the containers as shown in Figure 23. A compete SCT API users guide can be found on the clcsmail server at the URL:

http://www-clcs/project/syscontrol/redman/ ClassList.html

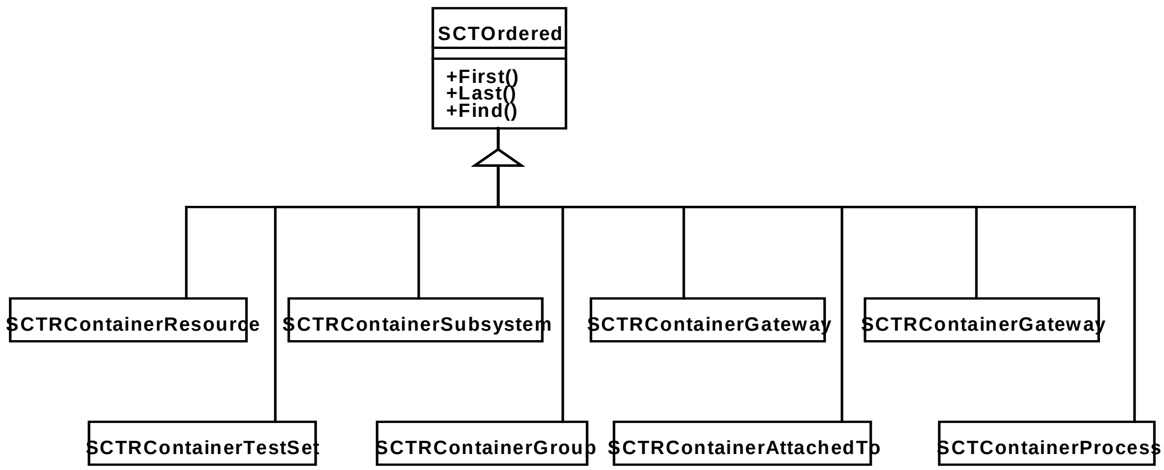


Figure 24. SCT Container Structure

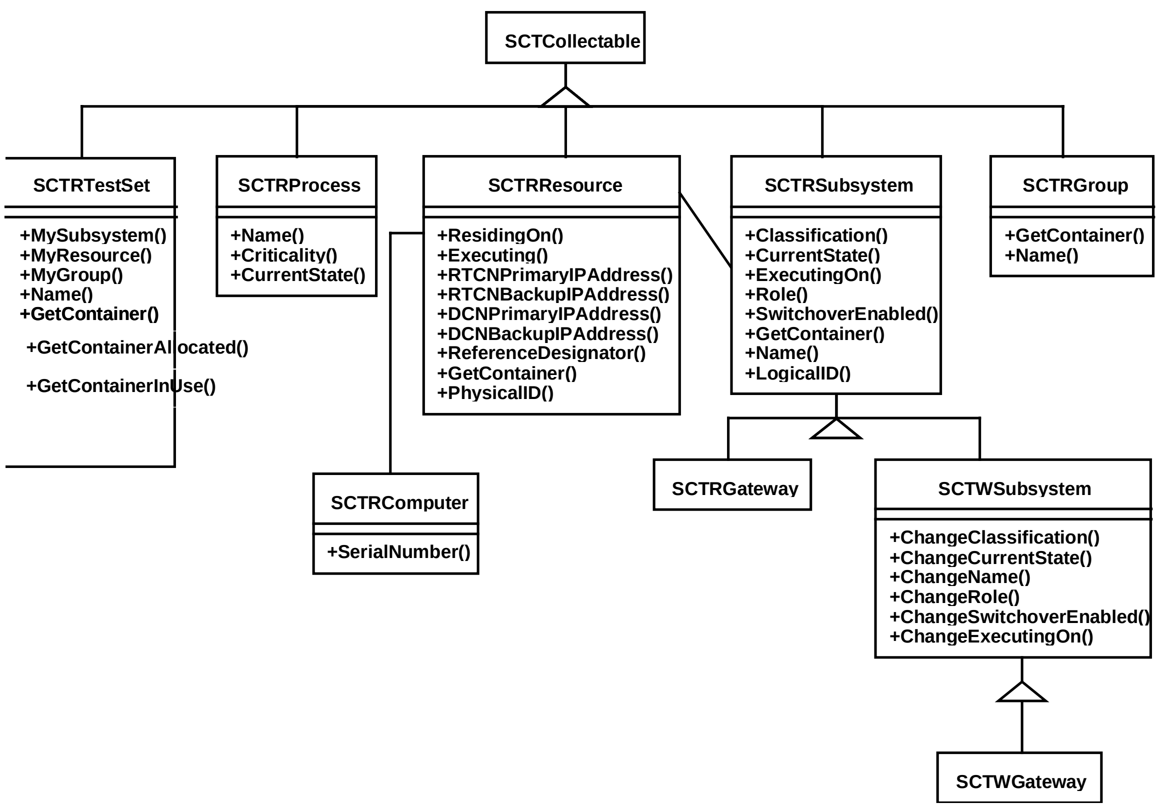


Figure 25. SCT Object Inheritance

1.1.5.16 System Configuration Table Table Formats

The SCT is built from 12 files. Each file is comma delimited and provides information that defines the part of the system configuration. The format of each of these files is described below.

1.1.5.16.1 Set File

This file specifies the name of the Set. This is static, but will vary from facility to facility. There is only one record in this file.
Name: sets.txt

Format: Comma delimited

Table VI. Set File Format

Field	Type	Contents
Set Name	String	This will be the Set Name.

Example:
IDE

1.1.5.16.2 Test Set File

This file specifies the Test Sets that can exist in the Set. This is expected to be used primarily by Set Integrity.
Name: testset.txt

Format: Comma delimited,

Table VII. Test Set File Format

Field	Type	Contents
Test Set Name	String	A unique name for a test set. There are no constraints on the contents of the string.

Example:
IDEA
IDEB

1.1.5.16.3 Group File

This file specifies the Groups of Resources that must be allocated to a single test set. This infomation is used primarily by Set Integrity to ensure that no group is split across test sets.
Name: groups.txt

Format: Comma delimited

Table VIII. Group File Format

Field	Type	Contents

Group Name	String	The Name of the group. This name is unique
Group Type	Integer	0 = Null Group 1 = Control Group 2 = Front End Zone Group 3 = Untitled Group

Example:
SME Gateways,2
VAB Gateways,1

1.1.5.16.4 Resource File

This file specifies the parameters for the individual resources in the Set.
Name: resource.txt

Format: Comma delimited

Table IX. Resource File Format

Field	Type	Contents
Host Name	String	The commonly used name for the Resource. This name is unique.
Physical ID	Int16	Number that uniquely identifies the resource within the set.
Reference Designator	String	This field specify the physical location of the resource
Primary RTCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the Primary RTCN. This field may be null (for CCWSs)
Backup RTCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the Backup RTCN. This field may be null (for CCWSs and all processors prior to installation of the backup RTCN).
Primary DCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the Primary DCN. This field may be null (for gateways)
Backup DCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the backup DCN. This field may be null (for gateways and all processors prior to the installation of the backup DCN)

Example:
IDE_CCP01,1,1A365,123.123.255.255,123.123.255.255,123.123.255.255,123.123.255.555
IDE_CCP02,2,1A366,233.233.233.233,233.233.233.233,233.233.233.233,233.223.223.223
IDE_DDP01,3,1A399,433.433.433.433,555.555.555.555,444.444.444.444,333.333.333.333
IDE_DDP02,4,1A400,433.433.433.433,111.111.111.111,222.222.222.222,333.333.333.333
IDE_GSE01,5,2A120,433.433.433.433,322.222.222.222,334.343.434.343,212.341.234.123
IDE_GSE02,6,2A250,344.344.344.344,333.333.333.333,123.412.341.234,234.123.412.341

1.1.5.16.5 Subsystem File

This file specifies the definition of the Subsystem
Name: subsys.txt

Format: Comma delimited

Table X. Subsystem File Format

Field	Type	Contents
Subsystem Name	String	The name of the subsystem being defined. Because this file is used by Set Integrity, a given subsystem name may appear more than once. The combination of Test Set/Subsystem Name is unique within the file.
Logical ID	Int16	Unique identifier for the subsystem. Used as the CPU ID in the network message header.
Switchover Enabled	Integer	When 0, Switchover is disabled. When 1 it is enabled. No other value is valid.
Role	Integer	0 = No Assigned Role 1 = Active 2 = Standby 3 = Hot Spare
Classification	Integer	The Type of Subsystem 0 = No Assigned Classification 1 = CCP 2 = Master CCP 3 = DDP 4 = CCWS 5 = Master CCWS 6 = Ops/CM 7 = Gateway

Example:
CCP1S,2,0,2,2
GSE1A,3,1,1,7

1.1.5.16.6 Subsystem to Test Set Map File

This file specifies the Test Set to which each Subsystem is allocated
Name: ss2ts.txt

Format: Comma delimited

Table XI. Subsystem to Test Set Map File Format

Field	Type	Contents
Test Set	String	Test Set Name. This will match a name provided in the test set file.
Subsystem	String	Subsystem Name. This will match a name provided in the subsystem file.
Subsystem Classification	Integer	The Type of Subsystem. This is consistent with the definition in the Subsystem File. 0 = No Assigned Classification 1 = CCP 2 = Master CCP 3 = DDP 4 = CCWS 5 = Master CCWS 6 = Ops/CM 7 = Gateway

Example:
IDEA,CCP1S,2

IDEA,GSE1A,7
IDEB,GSE1A,7

1.1.5.16.7 Process File

This file specifies the processes in the system.
Name: process.txt

Format: Comma delimited

Table XII. Process File Format

Field	Type	Contents
Process Name	String	A unique name for the process
Critical	Integer	0 = Not a Critical Process 1 = Critical Process

Example:
CCPP1,0
CCPP2,0

1.1.5.16.8 Resource to Group Map File

This file specifies the Resources contained in each identified group.
Name: res2grp.txt

Format: Comma delimited

Table XIII. Resource to Group Map File Format

Field	Type	Contents
Group Name	String	Name of the Group. This will be one of the groups specified int he Group File.
Resource Host Name	String	Name of the Resource allocated to the Group. This will be one of the resources specified in the Resource File. A given resource will only be assigned to one group.

Example:
SME Gateways,IDE_GSE01
SME Gateways,IDE_GSE02
VAB Gateways,IDE_GSE01

1.1.5.16.9 Process to Subsystem Map File

This file specifies the processes that make up each subsystem.
Name: proc2ss.txt

Format: Comma delimited

Table XIV. Process to Subsystem Map File Format

Field	Type	Contents
Subsystem Name	String	The Name of the Subsystem. This is one of the Subsystems identified in the Subsystem File
Process Name	String	The Name of the Process that is part of the subsystem. This is one

		of the Processes identified in the Process File.

Example:
CCP1S,CCPP1
CCP1S,CCPP2

1.1.5.16.10 Resource to Subsystem Map File

This file specifies the mapping between resources and subsystems.
Name: res2ss.txt

Format: Comma delimited

Table XV. Resorce to Subsystem Map File Format

Field	Type	Contents
Subsystem Name	String	The Subsystem Name. This is one of the Subsystems identified in the Subsystem File.
Resource Name	String	The Host Name of the resource. This is one of the resources identified in the Resource file.

Example:
CCP1S,IDE_CCP01
GSE1A,IDE_DDP01
GSE1A,IDE_GSE01

1.1.5.16.11 Resource to Test Set Map File

This file specifies the resources that have been allocated to the test sets. Allocation is defined to be that it may be used. A given resource can be allocated to multiple Test Sets. It can only be in use by one at any given time.
Name: res2ts.txt

Format: Comma delimited

Table XVI. Resource to Test Set Map File Format

Field	Type	Contents
Test Set Name	String	This matches one of the Test Sets identified in the Test Set File.
Resource Name	String	This is the Host Name of one of the resources identified in the Resource File.

Example:
IDEA,IDE_CCP01
IDEA,IDE_CCP02
IDEA,IDE_DDP01
IDEA,IDE_DDP02
IDEA,IDE_GSE01
IDEA,IDE_GSE02
IDEB,IDE_CCP02
IDEB,IDE_GSE01

1.1.5.16.12 Group to Test Set Map File

This file specifies the groups that have been allocated to the test sets. The group can be allocated to multiple test sets, but can be in use by only one test set at any given time.
Name: grp2ts.txt

Format: Comma delimited

Table XVII. Group to Test Set Map File Format

Field	Type	Contents
Test Set name	String	This is the name of one of the Test Sets specified in the Test Set File.
Group name	String	This is the name of one of the Groups specified in the Group File.

Example:
IDEA,HMF Gateways
IDEA,SME Gateways
IDEB,VAB Gateways

1.1.6 SCT Build

The SCT Build is a Microsoft Access Database. The Database Tables closely mirror the files built for the SCT (See Figure 26). As currently populated, a single database contains the data for all sets. The same design supports independent databases for each set, or some combination of the two. The tables can be grouped into two types: Data and Relationship. The data tables specify the values for each object in the SCT (subsystems, resources, test sets, etc.). The Relationship tables specify the relationships between objects (which subsystem executes on which resource, the resources that belong to a set, etc.). In order to minimize errors introduced at data entry, all relationship tables must contain values already defined in the Data Tables.

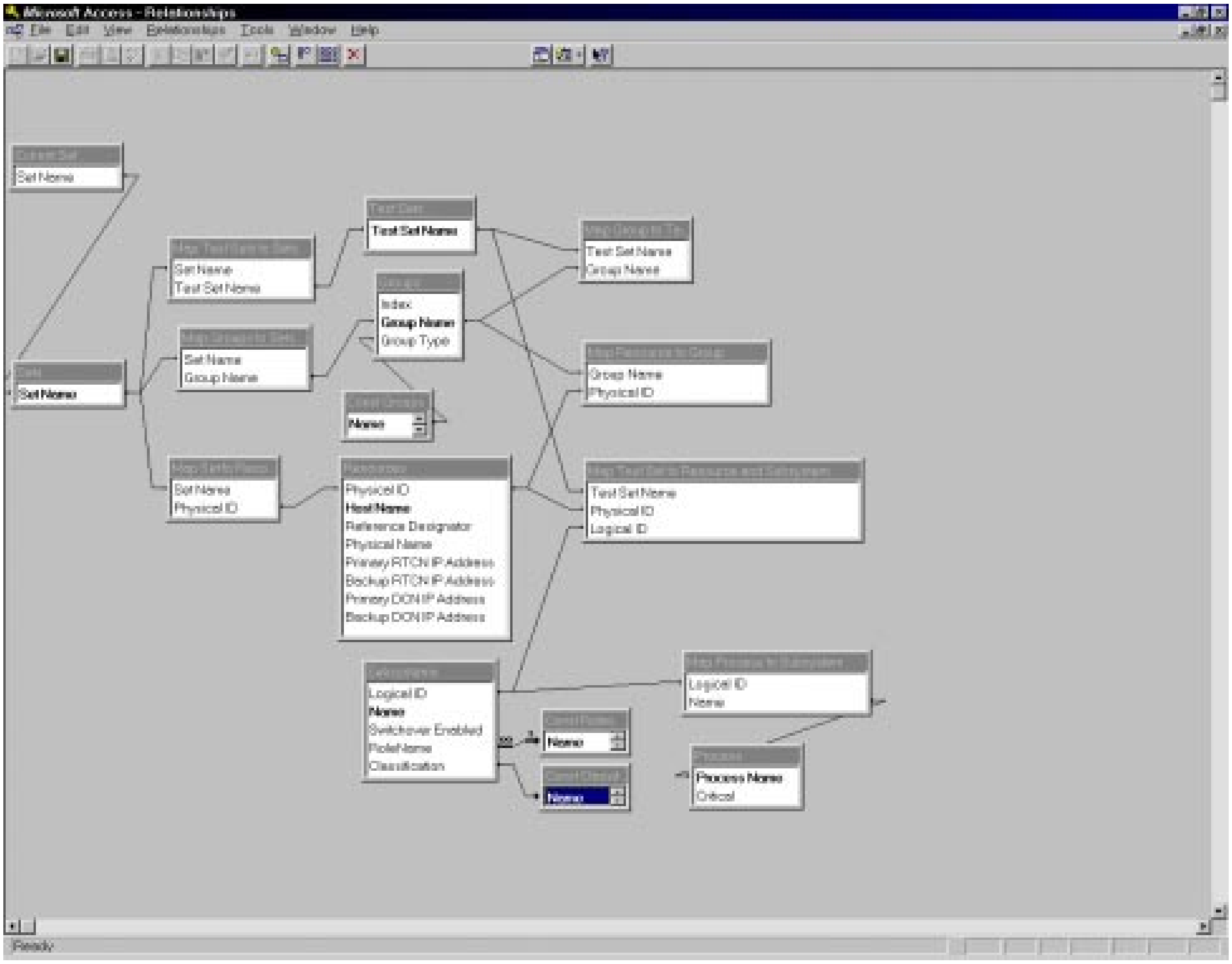


Figure 26. SCT Build Data Tables

1.1.6.1 SCT Build Detailed Data Flow

TBD

1.1.6.2 SCT Build System Context Diagram

TBD

1.1.6.3 SCT Build State Definition and State Transition Diagram

TBD

1.1.6.4 SCT Build Unique Algorithm Design

TBD

1.1.6.5 SCT Build Development Tools

TBD

1.1.6.6 SCT Build External Interfaces

TBD

1.1.6.7 SCT Build Data Dictionary

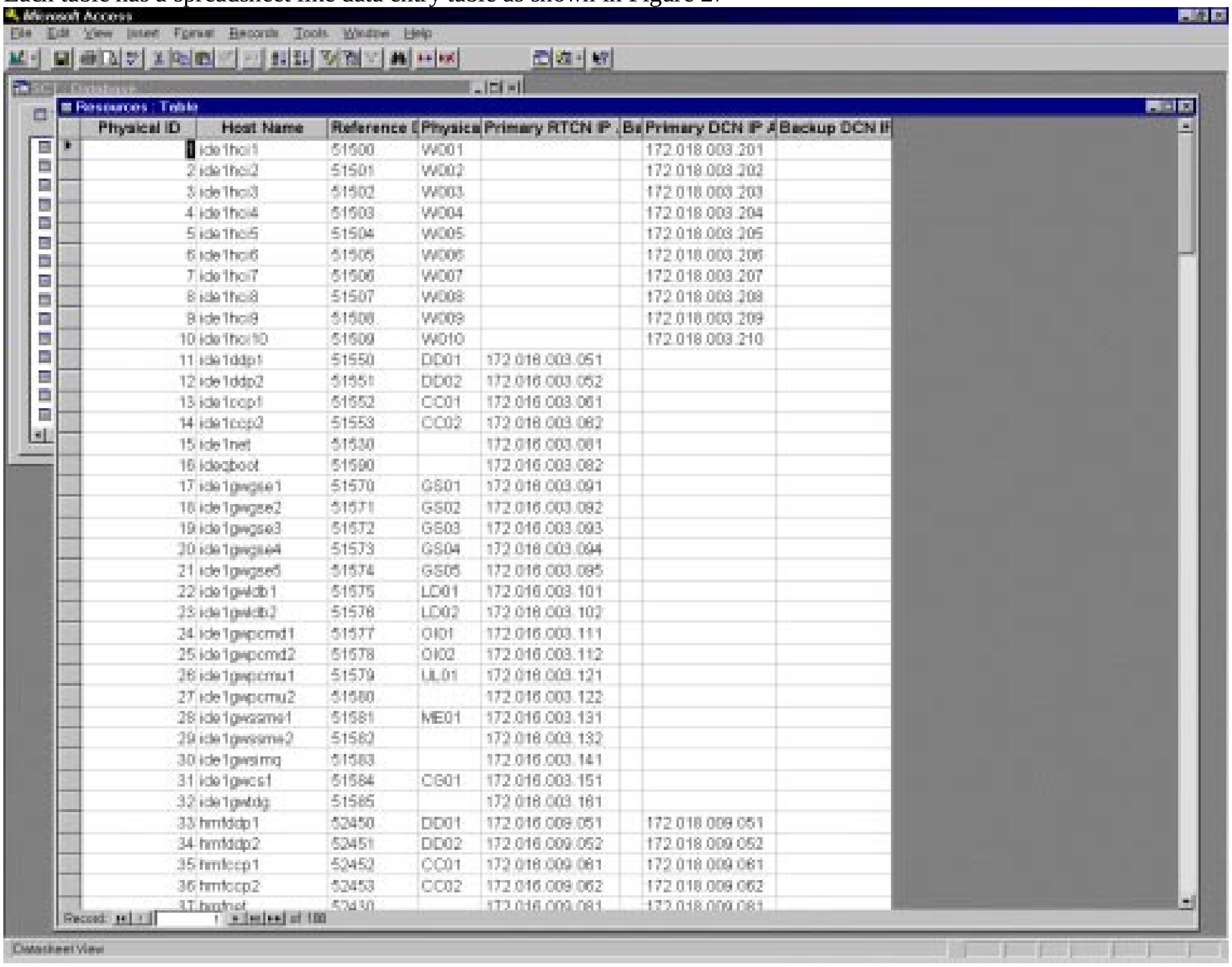
TBD

1.1.6.8 SCT Build Message Formats

SCT Build generates no system messages.

1.1.6.9 SCT Build Display Formats

Each table has a spreadsheet like data entry table as shown in Figure 27



Physical ID	Host Name	Reference [	Physica	Primary RTCN IP .	Ba	Primary DCN IP A	Backup DCN IF
1	ide1hci1	51500	W001			172.018.003.201	
2	ide1hci2	51501	W002			172.018.003.202	
3	ide1hci3	51502	W003			172.018.003.203	
4	ide1hci4	51503	W004			172.018.003.204	
5	ide1hci5	51504	W005			172.018.003.205	
6	ide1hci6	51505	W006			172.018.003.206	
7	ide1hci7	51506	W007			172.018.003.207	
8	ide1hci8	51507	W008			172.018.003.208	
9	ide1hci9	51508	W009			172.018.003.209	
10	ide1hci10	51509	W010			172.018.003.210	
11	ide1ddp1	51550	DD01	172.016.003.051			
12	ide1ddp2	51551	DD02	172.016.003.052			
13	ide1ccp1	51552	CC01	172.016.003.061			
14	ide1ccp2	51553	CC02	172.016.003.062			
15	ide1net	51530		172.016.003.081			
16	ideqboot	51590		172.016.003.082			
17	ide1gwgse1	51570	GS01	172.016.003.091			
18	ide1gwgse2	51571	GS02	172.016.003.092			
19	ide1gwgse3	51572	GS03	172.016.003.093			
20	ide1gwgse4	51573	GS04	172.016.003.094			
21	ide1gwgse5	51574	GS05	172.016.003.095			
22	ide1gwldb1	51575	LD01	172.016.003.101			
23	ide1gwldb2	51576	LD02	172.016.003.102			
24	ide1gwpcmd1	51577	OI01	172.016.003.111			
25	ide1gwpcmd2	51578	OI02	172.016.003.112			
26	ide1gwpcmu1	51579	UL01	172.016.003.121			
27	ide1gwpcmu2	51580		172.016.003.122			
28	ide1gwssme1	51581	ME01	172.016.003.131			
29	ide1gwssme2	51582		172.016.003.132			
30	ide1gwsimq	51583		172.016.003.141			
31	ide1gwcs1	51584	CS01	172.016.003.151			
32	ide1gwtdg	51585		172.016.003.161			
33	hmfddp1	52450	DD01	172.016.009.051		172.018.009.051	
34	hmfddp2	52451	DD02	172.016.009.052		172.018.009.052	
35	hmfccp1	52452	CC01	172.016.009.061		172.018.009.061	
36	hmfccp2	52453	CC02	172.016.009.062		172.018.009.062	
37	hmfnet	52430		172.016.009.081		172.018.009.081	

Record: 1 of 100

Figure 27. Resource Data Entry Table

Once all data for a set is correct, the set name is entered in the Current Set table and the Export SCT Macro is executed. This produces the files for the set which can then be made available to OPS/CM for download to each of the computers.

1.1.6.10 SCT Build Input Formats

TBD

1.1.6.11 SCT Build Recorded Data

TBD

1.1.6.12 SCT Build Local Storage Requirements and Formats

TBD

1.1.6.13 SCT Build Printer Formats

TBD

1.1.6.14 SCT Build Interprocess Communications (C-to-C Communications)

TBD

1.1.6.15 SCT Build External Interface Calls (e.g., API Calling Formats)

TBD

1.1.6.16 SCT Build Table Formats

The SCT Build generates files used by the online SCT. The specifications for these tables are in the SCT Design section of the document.

1.1.7 Redundancy Management Test Plan

1.1.7.1 Test Environment

The equipment necessary for the Thor 3.0 baseline test will include at a minimum:

1 CCP
2 DDP
3 CCWS
4 Ops/CM (optional)

1.1.7.2 Test Tools

In an effort to coordinate our testing efforts, this CSC can utilize the Ops/CM server in order to make the subsystem state transitions into GO as well as the registration of the processes executing on the platform. The Redundancy Management executables can be utilized to cycle through the process states. Stubs provided by Redundancy Management can also be used to achieve the same level of testing.

1.1.7.3 Test Plan

It is the aim of this test plan to demonstrate the intended functionality/capability of the Redundancy Management CSC as it pertains to the implementation of Thor 3.0, described within the Subsystem Integrity section of this document. These tests will add to the current functionality/capability to build upon for Atlas. The test cases are specified towards Subsystem Integrity, however due to the nature of the modifications necessary to implement Thor 3.0, the testing is not limited to Subsystem Integrity.

1 Demonstrate the ability to register and unregister a process both formally and informally while making the action known to all SCTs residing on those platforms participating in the test.

2 Demonstrate the ability for a process to transition through all required states while making the action known to all SCTs residing on those platforms participating in the test.
3 Demonstrate the ability for a process to report an error and have that error issued by SSI local to the platform where the error was reported.

1.2 Notes

1.2.1 System Event Codes

Table XVIII. System Event Codes

SEC Number	Name	Source	Destination
256	Subsystem Loaded	SSI	SI (Master SCT)
257	Subsystem Communicating	SSI	SI (Master SCT)
258	Subsystem Go	SSI	SI (Master SCT)
259	Subsystem NoGo	SSI	SI (Master SCT)
260	Not Communicating		
261	Subsystem Not Loaded	SSI	SI (Master SCT)
262	Terminate	SI	SSI on Targeted Platform
263	*Switchover Directive*	*SI*	*SSI on Targeted Platform*
264	*New Active*	*SI*	*All SSI (Local SCT)*
393	*Subsystem running ORT*	*SSI*	*SI*
394	Subsystem not running ORT	SSI	SI (Master SCT)
395	No Packet Received from GW	SI-DDP	SI-CCP
396	Standby GSE detected no poll from Active GSE	GSEnS	*SI*
397	GSE reports no response from bus	GSEnA	*SI*
398	HC not Incremented	SI-DDP	SI-CCP
399	HC has Decremented	SI-DDP	SI-CCP
400	Terminate Gracefully	SI	SSI on Targeted Platform
401	Subsystem Loaded	SI (Master SCT)	All SSI (Local SCT)
402	Subsystem Communicating	SI (Master SCT)	All SSI (Local SCT)
403	Subsystem in Go	SI (Master SCT)	All SSI (Local SCT)
404	Subsystem NoGo	SI (Master SCT)	All SSI (Local SCT)
405	Subsystem Not Communicating	SI (Master SCT)	All SSI (Local SCT)
406	Subsystem Not Loaded	SI (Master SCT)	All SSI (Local SCT)
408	CPU Utilization	CI	SI (Master SCT)
409	Available Memory	CI	SI (Master SCT)
410	Disk Utilization	CI	SI (Master SCT)
411	Disk Accesses	CI	SI (Master SCT)
412	Disk Errors	CI	SI (Master SCT)
413	Initial HC Received	SI-DDP	SI (Master SCT)

SEC Number	Name	Source	Destination
500	SCT Relinquish Request	Master SCT	Acting Master SCT
501	SCT Relinquish Response	Acting Master SCT	Master SCT
502	SCT Relinquish Accept	Master SCT	Acting Master SCT

1.2.2 Thor Process Table

Table XIX. Processes Generating Heartbeats to SSI

CSC	Process	Critical	Periodic	Rate
Redundancy Management	Low Rate	No	Yes	.1 Hz
Redundancy management	Event Driven	Yes	No	TBD
Data Distribution	High Rate	Yes	Yes	100 Hz
Timer Services	High Rate	Yes	Yes	100 Hz
TBD				

1.2.3 Message Stack

Table XX. Message Stack

				Command Processor Contact: Brian Hooker
				FD Commanding (FD Write Services) SW Required:None Data Required: CVT (can be stale), OLDB (fil Other HW/SS: None to get out of the box, but CCP, and DDP to deliver it Value Add: Standard interface for commandin Contact: Julia Samson
				Command Mangement API SW Required: None Data Required: SCT to build the packet Other HW/SS: None Value Add: Command Authentication
		System Event Codes SW Required: None Data Required: SCT Other HW/SS: None	**System Messages** SW Required: System Message Process Data Required: SCT, System Message Catalog Other HW/SS: Master CCP - Router Process, CCWS to see the message.	**Command Management** SW Required: Command Management Proces Data Required: SCT Other HW/SS: CCP in Go. For Psuedo FD d DDP must also be in go. For Gateway FDs, gateway must be GO, and activated. No dat available from gateway unless the DDP is (Value Add: Command Authentication Contact: Brian Hooker
CORBA* SW Required: ?? Data Required: ?? Other HW/SS: None Value Add: Location Transparency, COTS	**Application Messaging (AM)** SW Required: None is required. NRS is Optional and started by ??	Purpose: Quick/reliable notification of system events Contact: Ken Castner	Purpose: User Notification of significant system events Contact: Lynn Higgins	**IPC Service:** SW Required: IPC Process Data Required: None Other HW/SS: None Value Add: Provides single queue for both loc Contact: McMahon (Houston)
Contact: Bob McMahon	Data Required: None Other HW/SS: None Value Add: Thin layer on top of sockets Contact: Steve Davis	**Connectionless Messaging (CLM)** SW Required: None Data Required: None Other HW/SS: None Value Add: Contact: McMahon	**Reliable Multicast (RM)** SW Required: RM Process (requires SCT to Start) Data Required: Loaded SCT, Activity, Logical ID, Network Other HW/SS: None	Value Add: Automatic recording, Reliable del Contact: Steve Davis
TCP		**UDP**		

Software Required: None Data Required: None	Other HW/Subsystems: None Value Add:	Software Required: None	Data Required: None	Other Hardware/Subsystems: None

This is the last page of the Document